EFFICIENCY/IMPACT OF CORRECT WORD PREDICTION VS TYPING EFFICIENCY

[1] *Prof. Manju More,* [2] *P Gautam,* [3] *Niranjan Ajgaonkar,* [4] *Samuel Jonathan,* [5] *Madeeha Afreen,*
[1] *Assistant Professor,* [2][3][4] *Student, School of Computer Science and Engineering,* [5] *Electronics and Communication Engineering, Reva University*

Abstract.

Nowadays, AI and ML is the fastest-growing field in computer science. Word prediction software helps in reducing the number of keystrokes, based on context and frequency the next word is predicted. Based on the words previously used by the user, word prediction programs provide the user with a list of likely words. Prediction is necessary to understand the language, these algorithms are specifically used by users who have difficulty in writing due to spelling deficits.
This paper discusses the working application of word prediction algorithms like LSTM, N-gram, BK Trees, and skip-gram; And goes on to compare the efficiency of NLP Word Prediction versus that of traditional typing to establish the better or more efficient way to increase typing efficiency.

INDEX TERMS: LSTM, N-gram, B-K tree, traditional typing technique

1. INTRODUCTION

Interactions between human language and computers, NLP (Natural Language Processing) is a subfield of artificial intelligence, linguistics, and computer science. Used in various applications.

Fig. 1: Where NLP lies

One such example is when a person texts another person, a suggestion bar pops up trying to predict the next likely word that the user might want to use. As shown in fig.1. NLP includes both ML and AI. It mainly deals with the process of prediction. This paper contains a detailed description of the three popular word predicting algorithms:

- Long Short Term Memory Algorithm
- N-Gram Model Algorithm and
- Burkhard Keller Tree Algorithm.

And also,

- A Field Survey;

That proves once and for all, the better of the two: Traditional Typing vs Typing with the use of word suggestions and NLP techniques.

We have used these 3 language models to give readers a better understanding about the logic behind word prediction and sentence automation. The said models are widely used in almost all keyboard applications and mobile phones.

Proceeding this, we have included an interesting experiment on a wide variety of people. The survey inputs data such as, whether people are willing to use automated text submissions provided by their devices, or if they prefer to brute force their way through the construction of words and sentences typing it all without any help. Is it more efficient to use type without using suggestions? What is the future scope for word predicting algorithms in NLP? The result albeit a bit peculiar at first, does hold up upon further inspection.

2. LITERATURE REVIEW

In the comparative study: Andrew Pulver and Siwei Lyu have proposed a much responsive variant of LSTM, that is LSTWM. They have explained the working and architecture of LSTM, and their proposed variant have also been explained. They've compared with their model to that of LSTM. They've concluded with a modified LSTM architecture that outperforms LSTM in several cases. They found an optimal solution for the performance of one of RNN architecture, this could be enriched and used. [1]

Ching Y. Suen has mentioned about the n-gram statistics for Natural Language understanding. He has done the statistical analysis for various combination of letters. And have discussed about the different types of application of n-gram. He found that the n-gram statistics for the application is very useful tool when compared with dictionary method which requires much greater storage and computing time. He concluded that with over the time the machine intelligence in understanding the natural languages and text processing would be closer enough to that of human beings. [2]

3. LSTM (LONG SHORT-TERM MEMORY)

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture. It is used in the field of deep learning (DL). Unlike the standard feedforward neural networks, LSTM comprises feedback connections. It can process not only one but an entire sequence of data (like videos or long speech). As an example, LSTM applies to tasks

such as speech recognition, unsegmented, handwriting recognition, and abnormal detection in network traffic or intrusion detection systems. [3]
An LSTM network is a type of RNN, it learns dependence on historic data for a sequence prediction task. It's the feedback connections that lets LSTM learn about these historic dependencies. When we speak about a common LSTM cell. We could identify an input gate, a forget gate and an output gate. The flow of information in an LSTM model depends on or is controlled by the weights of these gates. And thus are the parameters that are learned during the training process. [4]
Some variants of LSTM model are Convolution LSTM (CNN – LSTM) and Bidirectional LSTM (Bi – LSTM). For the most part, these variants correspond to different encodings of the input sequence. A CNN- The result of using CNN as the encoding layer has certain cons, that is it needs many parameters, and even bigger model sizes were required. Which isn't fit for quite a few tasks. [4]

i. LSTM architecture:

LSTM deals with both Long-Term and Short-Term Memory. And it makes the calculations very simple and yet effective. [5] When we speak about the Vanishing gradient problem, it is faced when artificial neural network (ANN) is trained with backpropagation and gradient-based erudition. Where at every consecutive iteration while training the neural networks; It is notified which is proportional to the partial derivative of the error function to that of the current weight. [6] And this problem is faced mostly when we go for backpropagation. And Backpropagation is the process where RNN tries to go through the time to backpropagate via the network so that it could adjust its weights in order of error reduction in the network. Here it got the name "though time" because in RNN it would have to deal with the sequential data each and every time when it tries to go back it's equivalent to go back in time to the past. [7] This has a huge effect, and the weight update process is widely affected. Also, the model could turn out to have no purpose. And thus, we use LSTM, which has a hidden state and a memory cell with three known gates. Which is input, output, and forget gate. [8]
As you can see in the fig.2. The LSTM unit is comprising of a memory cell, an output gate, a forget gate, and an input gate. The cell can remember the values within the time intervals and also helps to regulate the flow of information throughout the three gates in the cell. The LSTM's design acquired its current state from the inspiration of the logic gates of the computer. The memory cell has the same shape as the hidden state. And the cell the gates are used. [9]

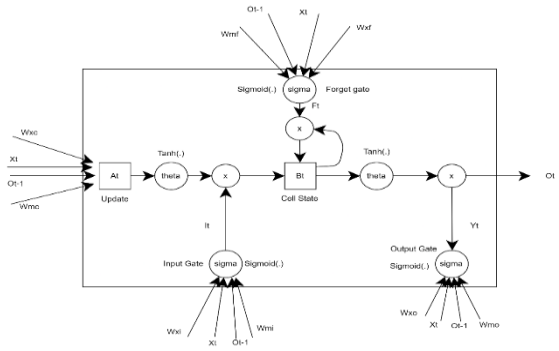

Fig.2: LSTM Unit

a. Forget gate

Forget gate is responsible for LSTM to remember, memorize and recognize all the data arriving in the network and to remove the data or information, which is not necessary for the network to learn the information and forecasts. Forget gate plays an important role in the decision on the passing of data via the layers of the networks. And here the input is of two types that which is expected by the network. One of the inputs requires the data from the previous layers. The other requires the data from the presentation layer. In the fig2, the data passes via the sigmoid function, and there the data has the tendency to fall down to zero and has a probability to be eliminated from the network. [10][11]

b. Input gate

This gate is playing a key role on decision on the importance of data by informing the cell state. As said above the forget gate figures out on eliminating the data from the input gate of the network. This decision is the quantity analysis of the importance of the data which would help the further layers to learn the data for forecasting the moves. Here the data passes via the sigmoid and hyperbolic tan functions. The sigmoid function makes the decision on the weight of data and hyperbolic tan function helps in reduction of the biased networks. [10][11]

c. Output gate

Output gate is the final gate of the circuit. It plays a key role on the decision for finding the following hidden state of the network. From where the data flows via the sigmoid function. Once the cell is updated, the cell state moved to the hyperbolic tan function, it later is multiplied by the sigmoid function of the output gate. This makes it easy for the hidden state to carry forward the data. [10][11]

d. Cell state

The data that is weight gained is calculated by the layer on the cell state when it passes via the cell state. Here the output of both gates, which is input, and output are made to multiply one another. The data that could have dropped out is then multiplied to near-zero values. [10][11]

ii. Mathematical representation.

Let's take that there could be *O* number of hidden units, let the bunch size be m and total number of input number be k. Thereby, the input would be Z_t $\in \in$ R^{m*k}. And in the preceding time step the hidden state would be O_{t-1} $\in \in$ R^{m*o}. And here the gates at time step t are put up. Input gate would be X_t $\in \in$ R^{n*h}, forget gate would be A_t $\in \in$ R^{n*h}, and output gate would be Y_t $\in \in$

R^{n*h}. They are furnished into the form shown below. [9]

$$\mathbf{X_t = (Z_t W_{xi} + O_{t-1} W_{ho} + b_i),}$$
$$\mathbf{A_t = (Z_t W_{xf} + O_{t-1} W_{hf} + b_f),}$$
$$\mathbf{Y_t = (Z_t W_{xy} + O_{t-1} W_{hy} + b_y),}$$

Where $\mathbf{W_{xi}}$, $\mathbf{W_{xf}}$, $\mathbf{W_{xy}}$

$\in \in$

R^{k*o} are weight variables or parameters and $\mathbf{b_t}$, $\mathbf{b_f}$, $\mathbf{b_y}$

$\in \in$

R^{1*o} are the biased variables or parameter. [9]

When talk about the memory cell, we could find a method that governs the input and forgetting. The two gates, which is the input gate X_t helps in governing the data that accounts through

$c\tilde{}c\sim$

$_t$ and the forget gate A_t checks the quantity of aged memory contents of the cell C_{t-1}

$\in \in$

R^{m*h} which is retained. And thus, we fall to an updated equation using the pointwise multiplication technique. [9]

$$C_t = A_t \cdot C_{t-1} + X_t \cdot c\tilde{}_t \quad \mathbf{Ct = At \cdot Ct - 1 + Xt \cdot c\sim t}$$

If the forget gate is mostly approximate to 1 and if also the input gate is mostly approximate to 0. Then the aged memory cells would be retained across the time and would be thrown to the current time interval. And hidden state, V_t is computed as the follows, and the interval is (-1,1) [9]

$$V_t = Y_t \cdot \tanh(C_t) \quad \mathbf{Vt = Yt \cdot tan hCt}$$

iii.Results

We've defined the model, here the LSTMs hidden state throw back an extra memory cell of the value 0 and the form of the bunch size and the total number of hidden units. [4][9]

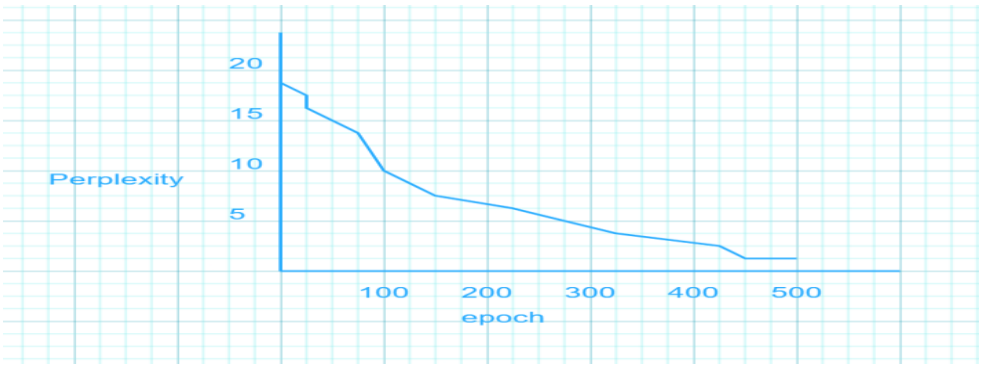

Fig.3: Output Graph of LSTM trained model

As shown in the fig.3 that once the model has been defined by providing all gates along with auxiliary memory cell. The output layer then receives the hidden state. The memory cell C_t has no direct significant role during the output computation. Once We train the LSTM that

is we define the function to train the model in one epoch. An epoch gives out the number of passes is done during the entire dataset training until the ML algorithm is completed.

4. N-GRAM LANGUAGE MODEL

If one has an array of words from a section, or a message that an individual is typing, an N-gram language model comes into play. This model has a specific but arduous task of completing a sentence typed by a user. It is never 100% accurate (unless mindreading was an attribute factored into it), but it still is competent enough to make a calculated prediction. An example of such a unigram set with predictions is – ("My","name","is"," SussySinwin"). Proper nouns are variable to the user.
This model learns the occurrences of selective words in a block of text, and then finds a train to correctly implement the next word as per the calculated probability.
A simple instance:
Let's take up two sentences-
"Career is my focus" and "Career is my thing." Going by a formal and general convention, the N-gram model will choose the former. The latter does not work well in a textual format and hence is omitted by the algorithm.

i. Basic Conventions and Calculations:
A unigram model only checks the incidence of a word. This is the first step towards building the model. The next step is the consideration of the prewritten sentence by the user, specifically known as the bigram model. As one moves forward by considering more words and sentences, the model eventually shapes up to be N-gram
Hence the probability for such a sentence would be calculated in this manner.

M('Entrepreneurship')
M('is '|'Entrepreneurship') P('my '|'Entrepreneurship is')
M('focus '|'Entrepreneurship is my')

This example can be iterated in another technique as well. Most notably, differentiating basic words from their own adjectives or adverbs. The complication of the English language containing multiple dialects is also detected by this model.

ii. Chain rule of probability:
$CR(O_1, O_2, ..., O_n) = CR(O_1|O_2, ..., O_n)\ CR(O_2|O_3, ...,O_n)\ \ CR(O_{n-1}|O_n)\ \ CR(O_n)$
$CR(w1,w2,w3\ldots wn) = \pi CR(wi|w1,w2..wn)$

Simplifying this using a Markov assumption we get 2 formulae for unigram and bigram.
Unigram: - $CR(w1,w2..wn)) = \pi CR(wi)$
Bigram:- $CR(wi|w1,w2..wi-1) = CR(wi|wi-1)$
Evaluation of an N-gram model
Intrinsic evaluation focuses on evaluating independent of the application. This method is a quick and versatile step for algorithm performance check.
To see how efficiently a model designs a prediction(perplexity) a relation is shown
Prob= w1w2..wn, perp (prob)=$CR\ (w1w2..wn)^{-1/N}$
This shows that perplexity is an inverse function of probability
The following graph in fig.4 shows a comparison between various n-grams normalized by words.

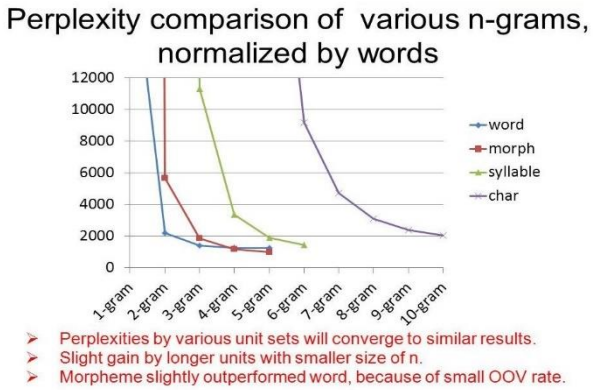

Fig.4 : Comparison of n-grams normalization by words

Perplexity of a set helps us compare various N-gram notations and models.

Language Modelling Metric

Entropy: $H(p)= \Sigma_x p(x).(-\log p(x))$

H(p)>=0. There exists a property, which calculates the plausibility of a distribution being accurate, know as cross entropy

This formula is used for calculation of probability of the test set assigned by the language model.

$perp(prob)=n\sqrt{\pi^{N}_{i=1}} 1/CR(wi|wi-1)$

This is a carefully chosen sentence for such a scenario: 'I am a God'. The first word prediction may go down in such a manner :

Word	**M ()**
A	0.4
God	0.3
I	0.12
Am	0.18

Probability of getting 'God' after 'a'

Word	**M ()**
A	0.05
God	0.3
I	0.15
Am	0.5

Probability of getting 'a' after 'am'

Word	**M ()**
A	0.1
Am	0.7
I	0.1
God	0.1

(These probabilities are taken on a curve)
But the general idea pf calculating perplexity remains the same
Hence from this data, we get the perplexity as
MM (W)=2.876 (equivalent)

iii.Inference:

This is a generic mathematical computation that is performed by an N-gram data model. This model needs to be trained on blocks of text/paragraphs for it to function smoothly.

A program (mostly written in python) can perform this task efficiently. But first, it must be trained on a sample. This is the N-gram model might not work as intended. A model that has been trained on the works of medieval poets will not return an accurate prediction if applied to other datasets. Out of vocabulary problems also occur due to this, where a word that does not appear during training appears during the testing.

5. BURKHARD-KELLER TREE:

A Levenshtein Distance is a language metric that is a building block for most algorithms. There exists an algorithm which does exactly that, known as the BK Tree algorithm. This structure is used for appropriate string matching using a dictionary.
This algorithm was adapted to discrete metric spaces (a function that defines a concept of distance between any two members of a set).
Case: **Discrete Metric d(x, y)**. Select an element 'x' as a root node of a tree. This node may have one or more subtrees. The nth subtree is recursively built of all elements 'y' such that d(x, y) = n. [12]
A list of words that need to be verified require a dictionary or a substantial reference point. To achieve this, an accurate reference, possibly a dictionary is used, for string/character matching.
Now if a user typed the word 'ar', the algorithm must then return a dictionary set with words that contain the same/similar letters.
Ex: {air, army}

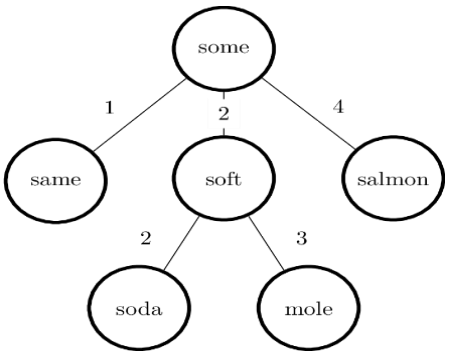

Fig.5 : Example of B-K Tree

As shown in the fig.5. This is a clear example of how BK Tree traverse through nodes of a root words. Every time a new word is added to the dictionary, it sprouts out from a child node of the root node. This child node then ends up becoming a new root node for the sub nodes.

The numbers indicate the number of changes that have been processed on the root word itself. Ex: someàsame has a value of 1, because only one letter has been edited.

A special value has been reserved for checking the number of edits made from the entered word to the theoretically correct word. This approach requires the algorithm to check the whole dataset consisting of all preexisting words, which returns an unnecessarily high complexity. Taking a case where a user has misspelled sola for the same tree.

First checkà. The root node is chosen and then the tolerance is calculated D("sola" -> "some") =4. Each node then branches out to a few child nodes. This child node is then put into the iteration function

Second checkà The second word used in this tree is soft. D("sola"," soft")=4. The tolerance factor is the same as above hence this word is not appended into the dictionary

Third Checkà We have reached the words soda and mole. Here D("sola," soda") =2, and D("sola"," mole") =4. Thus the control shifts to the soda node and attaches that to the dictionary list, i.e {"soda"}. For a bigger tree, with greater specific values the dictionary would expand. Thus, giving a wider range of selection of words. [13]

i.Insertion algorithm in a BK tree

The insertion primitive used to populate a BK tree f according to a metric d

- INPUT:

f: BK Tree;
d(u,v) for an arc u,v
w_u word assigned to the node u
d: metric used by f
w: element inserted into f

- OUTPUT:

Node off corresponding to w

- ALGORITHM:

If f is empty:
Create a root node r in f
w_r ß w
return r
Set up to the root of f
While u exists:
Kßd(w_u, w)
If k=0:
Return u
Find v i.e child of u such that d(u,v)=k
If v is not found:
Create the node v
w_v ß w
create arc(u,v)
d(u,v)ßk
return v
ußv

ii.Lookup algorithm in a BK tree

- INPUT:

dmax: maximum distance allowed between the best match and w.
(The rest is the same as above)

- OUTPUT:

w_{best}: the closest element to w stored in f according to d or Φ

- ALGORITHM:

If {\displaystyle t}t is empty:
Return Φ {\displaystyle \perp }
Create {\displaystyle S}S a set of nodes to process and insert the root of {\displaystyle t}t into {\displaystyle S}S.
{\displaystyle (w_{best},d_{best})\leftarrow (\perp ,d_{max})} (w_{best}, d_{best})ß(Φ,dmax)
While S != Φ {\displaystyle S\neq \emptyset }:
Pop an arbitrary node {\displaystyle u}u from {\displaystyle S}S
{\displaystyle d_{u}\leftarrow d(w,w_{u})}d_u ß d(w, w_u)
If d_u < d_{best} {\displaystyle d_{u}<d_{best}}:
(w_{best}, d_{best})ß(w_u, d_u){\displaystyle (w_{best},d_{best})\leftarrow (w_{u},d_{u})}
For each egress-arc(u,v) {\displaystyle (u,v)}:
If {\displaystyle |d_{uv}-d_{u}|<d_{best}}|d_{uv}-d_u|<d_{best}: (cut-off criterion)
Insert {\displaystyle v}v into {\displaystyle S}S.
Return w_{best} {\displaystyle w_{best}}

The output should be enough to suggest the working of any search algorithm however, to understand the mechanics of BK tree refer the algorithm stated earlier while reading the program (Algorithm)

6. FIELD SURVEY

Word suggestions is a technique to speed up text input by predicting what users are typing before they do, and suggest shortcuts, so they do not have to type the entirety of a word. Word suggestions are widespread they are available on virtually every mobile device but also on laptop computers. Millions of users are daily exposed to word suggestions. Still, we actually know quite little about how they are used, and how they can benefit users. Previous work even showed they can in fact slow down users. But where do we move from here? Does this mean suggestions are not accurate enough? How accurate do they need to be? Does it depend on how fast users can type? To begin with, for example the device they use? In a nutshell, what is the impact of accuracy and typing on the usage and benefits to entry speed of word suggestions?
To answer this, we designed a simple transcription task along with an attached survey. The device used to type was used as a proxy for typing efficiency. The proposed method considers Words Per Minute or WPM as the metric for measuring a person's typing efficiency. The resultant WPM will then be used to compare across different types of devices and depending upon whether or not the participants used the word predictions made available on their devices, we will be able to establish a winner between the two techniques. A sample screenshot of the WPM data collected is available here [15].

Here is what the different conditions looked like

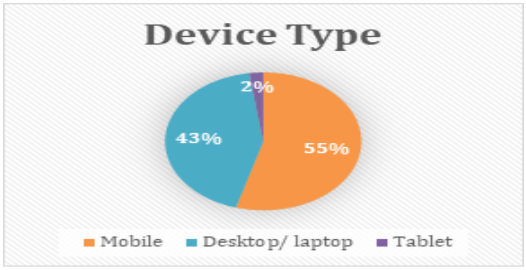

Fig.7: Devices that people used percentage

Out of 50 participants in the survey, roughly 55% were mobile users, 43% were desktop or laptop users and only 2% were tablet users.
Next, we asked our participants how often they used word suggestions available on their devices. fig.8

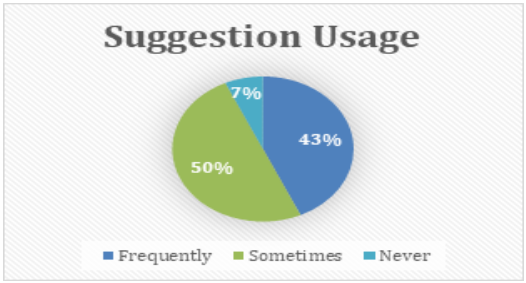

Fig.8: The percentage of how much people used predicted word

We found that 43% of our participants used word suggestions on a regular basis, 50% used this feature sometimes and only 7% never used this feature. After this, participants were asked to complete a 30 second transcription task created on the popular typing website, monkeytype.com, [16] that measured efficiency and accuracy of the users as they keyed in the given word-set.[15]

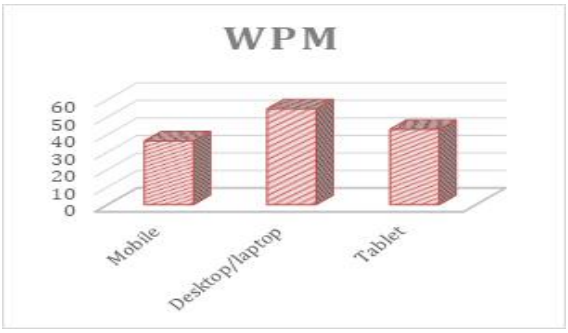

Fig.9: Average typing efficiency of three devices

Among the three device types, you can see in fig.9 that the average typing efficiency among mobile users was found to be 36.33 WPM (Words Per Minute), desktop or laptop users scored 54.68 WPM and tablet users had a score of 43 WPM.

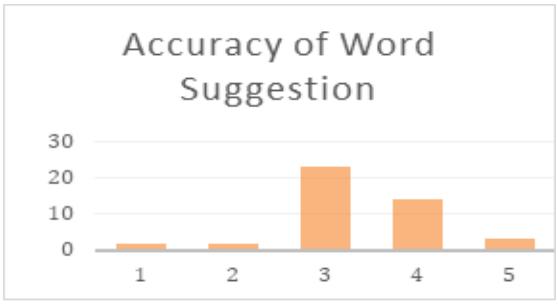

Fig.10: Accuracy of suggested word

We asked our participants how accurate the words were suggested to them, in their daily use. We included two factors, the accuracy of word suggestions, ranging from 1 - upsettingly inaccurate to 5 - uncannily accurate, and the device used to type. It was found that participants did not use much suggestions, except when they were extremely accurate. In fact, virtually no suggestions were used on laptop/ desktop devices.
Lastly, we asked the participants which of the two typing methods they thought was more efficient. fig.11

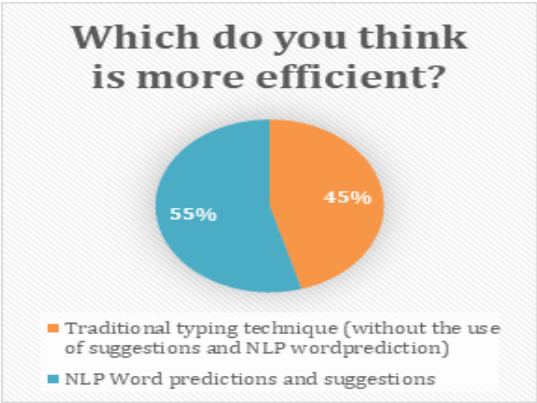

Fig.11: Response of the efficiency between Traditional tying and NLP word prediction

55% said that Word predictions and suggestions are more efficient while 45% said that Traditional typing was the better of the two. In this case the majority were wrong. Comparing the typing efficiency and accuracy of our participants from the transcription task earlier fig.9, we saw that Desktop users, who did not make use of word suggestions were most efficient, compared to the other two types of users.

7. CONCLUSION

To conclude, word suggestions must be uncannily accurate to improve entry speed. How beneficial they are depends upon how fast users can type. If we want to improve entry speed, designing faster typing techniques appears more effective than improving the accuracy of word suggestions. In the present scenario, training humans to type easily beats the efficiency of any word predicting algorithm.

In the future, for an algorithm to compete with the traditional typing technique, its accuracy must be at least 90% consistently, where 90% accuracy feels like the system is reading your mind. This will likely not be accomplished in the immediate future as training an algorithm to read a person's mind with such accuracy will take an indefinite amount of time.

8. REFERENCES

1. https://ieeexplore.ieee.org/document/7965940
2. https://ieeexplore.ieee.org/document/4766902
3. https://en.wikipedia.org/wiki/Long_short-term_memory
4. https://medium.com/linagoralabs/next-word-prediction-a-complete-guide_d2e69a7a09e6
5. Singh, K. D., & Ahmed, S. T. (2020, July). Systematic Linear Word String Recognition and Evaluation Technique. In *2020 International Conference on Communication and Signal Processing (ICCSP)* (pp. 0545-0548). IEEE.

5. https://www.analyticsvidhya.com/blog/2021/01/understanding-architecture-of-lstm/
6. https://en.wikipedia.org/wiki/Vanishing_gradient_problem
7. https://www.analyticsvidhya.com/blog/2021/07/lets-understand-the-problems-with-recurrent-neural-networks/
8. https://www.analyticsvidhya.com/blog/2021/08/predict-the-next-word-of-your-text-using-long-short-term-memory-lstm/
9. https://d2l.ai/chapter_recurrent-modern/lstm.html
10. https://analyticsindiamag.com/a-complete-guide-to-lstm-architecture-and-its-use-in-text-classification/
11. https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/
12. https://www.tutorialspoint.com/bk-tree-introduction-in-cplusplus
13. https://www.geeksforgeeks.org/bk-tree-introduction-implementation/
14. https://en.wikipedia.org/wiki/BK-tree
15. https://drive.google.com/file/d/1VxzSNcWYXmd8XgWss14V1LeLBc6GEXgw/view?usp=sharing
16. https://monkeytype.com/