Chapter 2

Methodology for Code Completion in Visual Codes for Scratch3 files

The methodology is described in four steps for next token prediction for Scratch3 [1] Visual codes files.

- Extracting Scratch3 [1] revisions file paths
- Splitting training and testing dataset
- Training dataset on KenLM [2], nltk [3] and BiLSTM [4]
- Evaluation of model

2.1 Extracting Scratch3 Revisions file paths

Revisions are changes made on a file over its lifespan either to correct errors, apply new methods or introduce new sets of components or data. [5] These changes when managed by a version control system such as git [6] contain relevant metadata such as *commitsha;* a unique id used for retrieving your contents stored in a git database [6], *commit messages* describing what changes you introduced and timestamp of the change. The Scratch3 [1] revision files was extracted from GitHub [7] using similar methodology adopted in the paper by *Eng et.al* on Identifying Defect-Inducing Changes in Visual Code. The revision files were parsed through our Scratch Parser

[8] to transform it into an Abstract Syntax tree [9] which is the intermediary representation for each file revision saved in our database. We utilized the revision file to generate a unique representation in hexadecimal stored in a hexadecimal digit format making it easy to distinctly and efficiently access the revision files. This led to creation of two columns in our content table. The hash field represents the hashed value of the content and the content field stored as a JSON [10] string containing the tree representation of the Scratch3 [1] revision file, its nodes count, edges count, list of connections and common occurrence of nodes and edges in the tree. The database content table contains the hashes; an md5 representation of the parsed revision file along with the original contents saved as JSON [10] string value. We utilized the hashes to retrieve its matching contents; a JSON [10] string value deserialized to a dictionary to access the connections of each specific Scratch3 [1] revision file. The connections were generated by recursively walking through the entire tree starting from the root to every node, we mark nodes visited and add it to the list forming a progressive order of nodes. This connection is a nested list with each element list containing a connection from the root node to every node in the tree in a progressive manner. For instance, if we have a tree of an Scratch3 revision file [1] shown below.This connection is a nested list with each element list containing a connection from the root node to every node in the tree.

Listing 2.1: Abstract Syntax Tree of a Simple Scratch3 File generated internally using our Scratch3 parser

```
[
        "simple_test",
        [
            [
                "event_whenflagclicked",
                [
                    [
                        "motion_movesteps",
                        [
                            [
                                "STEPS",
                                [
```

```
                    "10"
                ]
            ]
        ]
    ]
]
```

We recursively visit every node from the root to each node in the tree to form a list of connections so as to form an order of nodes suitable for creating an ngram model [11] for training.

Listing 2.2: Connections from a Simple Scratch3 File

```
[
        [
            "simple_test",
            "event_whenflagclicked"
        ],
        [
            "simple_test",
            "event_whenflagclicked",
            "motion_movesteps"
        ],
        [
            "simple_test",
            "event_whenflagclicked",
            "motion_movesteps",
            "STEPS"
        ],
        [
            "simple_test",
            "event_whenflagclicked",
            "motion_movesteps",
            "STEPS",
            "10"
        ]
    ]
```

The parsed tree was generated from the blocks dictionary in our Scratch3 revision JSON file [1] object which contains a plethora of relevant visual code information such as *events blocks;* the major control system blocks, *motion blocks;* useful for moving

the Sprite around, *looks blocks;* helping creatives display plain or animated messages and costumes, *control blocks* for switching and decision making. Other blocks that introduce some level of extra control for the users are the *MyBlocks*. This block is referred to as procedures within the Scratch3 JSON [10] Object enabling Scratch creatives generate a customized block. The parser captures every detail on the visual code information making it robust for the task ahead.

The extracted connections were further pre-processed by striping the first element of each connection which is the title of the Scratch3 revision file [1] so as to keep the connections as only visual code nodes. The next step involved normalizing the connections by replacing the non visual codes such as regular values like Hello, 10 and other values that are specifically entered by the user and replacing them with ⟨UserInput⟩. This was finally extracted into a different text file for each Scratch3 revision file [1].

2.2 Splitting Training and Testing Dataset

The projects were split into a training and test set using the split train_test_split method from scikitlearn [12] python library into 80% training set and 20% test set. This was necessary to ensure that testing is done on a new dataset on a project it hasn't seen before. The hashes corresponding to the training set parsed revision files and testing set parsed revisions files were compared and similarities found in the test hashes were discarded to ensure that test data were a unique set of data. The process from the previous steps led to the extraction of ⟨x⟩connections for the training set and ⟨y ⟩connections for the test set. The training data was padded with ⟨s ⟩token corresponding to the ngram [11] order employed for training. For instance if we are to train with bigrams we padded the paths with two ⟨s ⟩tokens at the start and end to ensure consistency.

2.2.1 Training Dataset on NLTK Language

We applied tokenization on the training set using the word_tokenize[13] method from the nltk [3] module to enhance analysis and training. In order to account for edge cases where preceding tokens might be insufficient for context, we employed the nltk padded_everygram_pipeline[14] to pad the dataset matching the order of ngram[11] we intend to train.The result of this along with the training data is passed onto the MLE [15] fit method to determine the maximum likelihood estimates of next visual code token. The model was saved in a pickle file to be used for evaluation as would be described in the evaluation steps.

2.2.2 Training Dataset on Kenlm

I utilized the KenLM[2] model on our training dataset. The approach as described in the official documentation was adopted for training and responses saved in an arpa [16] file. In more details, the Scratch3 [1] corpus which is sent through the stdin to the modified KneserNey smoothing [17] for estimation. Additionally, parameters such as the ngram [11] order and discount fallback were parsed for the estimation. This is sent to an arpa [16] file used for evaluation.

2.2.3 Training Dataset on BiLSTM

Similar to the nltk [3] library, tokenization was applied to divide each sentence in our dataset into tokens. In essence every visual code sentence is split into separate tokens. For instance if we have a sentence such as "control_start_as_clone sound_play SOUND_MENU sound_sounds_menu", with tokenization we can have it split into the following tokens.

- control_start_as_clone
- sound_play
- SOUND_MENU

- sound_sounds_menu

Furthermore, we created a vocabulary from the token where a unique index is assigned to each token in the corpus. We applied padding on these tokens using the tensorflow pad_sequence [19] function to make it consistent when fed into the model for training. To achieve this padding, we compared each sentence in our corpus with every other sentence and determined the sentence with the maximum length. This becomes the benchmark for padding as every other sentence in our corpus is padded to match the length of the maximum sentence in our corpus. Subsequently the last token of each sentence sequence is set to be the expected outcome. Considering a sentence sequence with n tokens, the features fed into the model would be the length of the sequence in this case n minus the last token. The last token becomes the output the model should predict based on the n - 1 tokens. Finally our data is ready for training. We applied some of these hyperparameters for our training as displayed in the table below and utilized the Tensorflow fit [19] method for training our dataset for 50 epochs.

Table 2.1: Hyper parameters used for Training on bilstm

Hyper parameters	**Values**
Learning Rate	0.01
Epochs	50
Activation Type	softmax
Optimizer	adam
LSTM	150

2.3 Evaluation of Models

The evaluation was carried out on the various models to ascertain how accurately it could predict the next visual code token. The procedure adopted for evaluating

the nltk[3] model is akin to the kenlm [2] model. In general the evaluation was done using a different dataset the model hasnt seen before to ensure an efficient model. The steps for evaluating each model are described in details below.

2.3.1 Evaluation of NLTK Model

The evaluation was carried out on the test dataset. We employed the context as all preceding tokens excluding the last token. The last token was the ground truth token adopted as the accurate predicted token. The nltk.lm score [3] method was adopted to estimate the log likelihood score of each prospect token sequence from the trained data alongside the context from the test dataset. All the prospect tokens scores were stored in a python dictionary [0] and the python max function was used to calculate the maximum value from all the scores of the prospect tokens. This value was selected and compared with the ground truth predicted token in our case the last token. This was computed and results were compared for accuracy, precision, recall and f1 score of the model. Fig 1 shows the ngram model [11] of order 2 to 8 compared with ngram [11] model of order 9 - 15. We adopted the Wilconxon non-parametric test [0] to statistically compare the metrics. We set up a Null Hypothesis [0] and Alternative Hypothesis [0] as listed below.

- **Null Hypothesis for NLTK:** There's no significant difference between the accuracy,precision,f1 and recall score value for nltk [3] model on my dataset for ngrams [11] of order 2 to 8 versus 9 to 15.
- **Alternative Hypothesis for NLTK:** There's a significant difference between the accuracy,precision,f1 and recall score value for nltk [3] model on my dataset for ngrams [11] of order 2 to 8 versus 9 to 15.

Our results show a p-value [0] score of 0.015625 across all our metrics; precision, recall,accuracy and f1 score [12]. This led us to reject our Null Hypothesis [0] that there's no significant difference between the accuracy, precision,recall and f1 score of

the ngram [11] model of order 2 - 8 versus 9 - 15. The plot in Fig1 which shows the metrics for ngram [11] model of order 2 - 8, compared with Fig2 which shows ngram [11] model of order 9 - 15. We see a significant difference in all our metrics across these two plots. We have the highest accuracy seen in the 10gram with an accuracy of 0.245. After 10-gram order the model maintained similar accuracy as no significant increase in accuracy was observed. The highest precision and f1 score was seen in the 10-gram with a score of 0.226 and 0.2176 respectively.

2.3.2 Evaluation of KenLM Model

The evaluation of the KenLM [2] model was carried out on the ARPA [16] file created during the training process. We utilized the arpa [16] file to create a vocabulary consisting of unique tokens which are mostly the tokens that appear under the one grams. Each line in the test data excluding the last token was concatenated with each word in the vocabulary to ascertain the token from the vocabulary that results in the maximum score chosen as the next predicted token. We adopted similar metrics used for nltk [3] ngram model to compute for accuracy, precision, recall and f1 score of our model performance across ngram [11] order 2 - 8 versus ngram [11] model of order 9 - 15. We adopted the Wilconxon nonparametric test [0] to statistically compare the metrics. We set up a Null Hypothesis [0] and Alternative Hypothesis [0] as listed below.

- **Null Hypothesis for KenLM:** There's no significant difference between the accuracy,precision,f1 and recall score value for KenLM [2] model on my dataset for ngrams [11] of order 2 to 8 versus 9 to 15.
- **Alternative Hypothesis for KenLM:** There's a significant difference between the accuracy,precision,f1 and recall score value for KenLM [2] model on my dataset for ngrams [11] of order 2 to 8 versus 9 to 15.

We got a p-value score [0] of 0.578125 for the accuracy and recall, and a p-value

score [0] of 0.8125 for f1 scores; hence we failed to reject the Null Hypothesis. In essence we didn't find a difference in the accuracy , recall and f1 score between the ngram [11] model of order 2 - 8 versus 9 - 15. On the other hand, we reject the Null Hypothesis for the precision score between ngram [11] model of order 2 - 8 versus 9 - 15 because of the p-value [0] of 0.015625. This tells us that there's a difference in precision between ngram [11] model of order 2 - 8 versus 9 - 15. Figure3 shows the result of evaluating the ngrams [11] model of order 2 to 8 with the highest accuracy and recall score attained in order 5 at 0.577. On the other hand the model had the peak precision in order 4 with a score of 0.762 and had the highest f1 score of 0.539 similar to accuracy and recall in order 5.

2.3.3 Evaluation of BiLSTM model

The predict function from the tensorflow.keras.models python module [19] was utilized for evaluation of our model. Similar to the nltk [3] and KenLM [2] model, we utilized the test data by choosing all the tokens in a sentence excluding the last token as context. This was transformed to sequences using the tensorflow Tokenizer texts_to_sequences method [19]. We padded the value adopting the same padding approach we used during the training phase by comparing each sentence against every other sentence in our corpus to ascertain the sentence with the maximum length. We employed the sklearn.metrics [12] module and adopted similar metrics such as accuracy_score, precision_score, recall_score and f1_score to get the metrics results for the BiLSTM model on 50 epochs.

Iĺl update the results

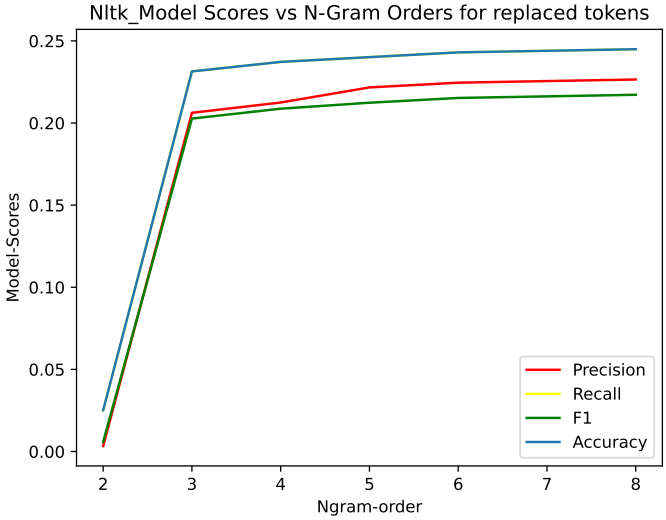

Figure 2.1: Nltk Model Scores vs N-Gram Orders 2 - 8

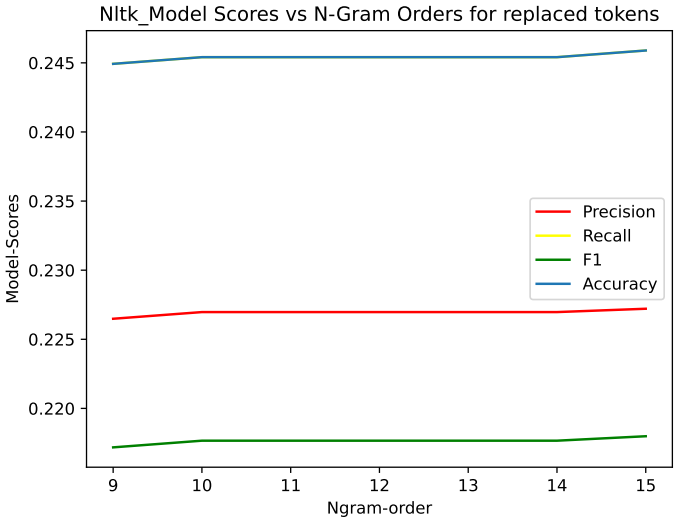

Figure 2.2: Nltk Model Scores vs NGram Orders 9 - 15

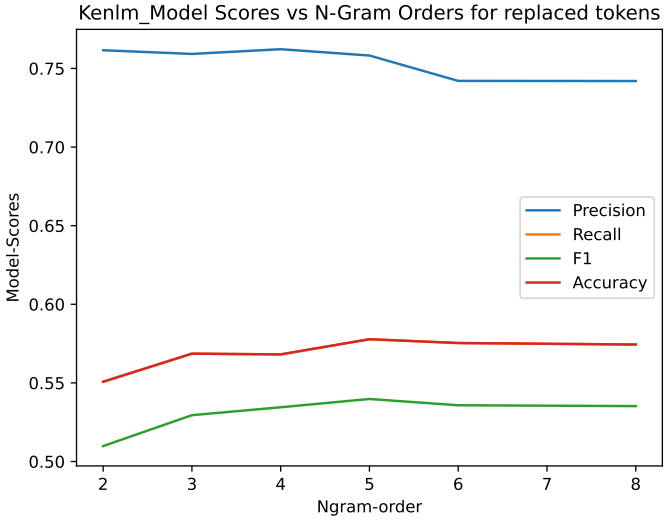

Figure 2.3: KenLM_Model Scores vs N-Gram Orders 2 - 8

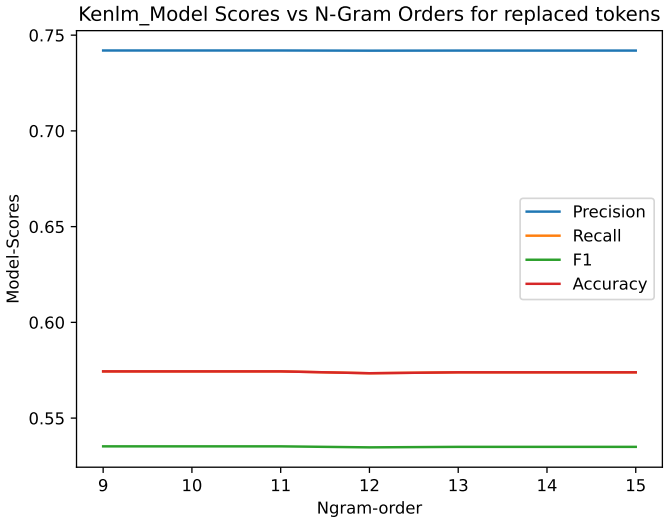

Figure 2.4: KenLM_Model Scores vs NGram Orders 9 - 15

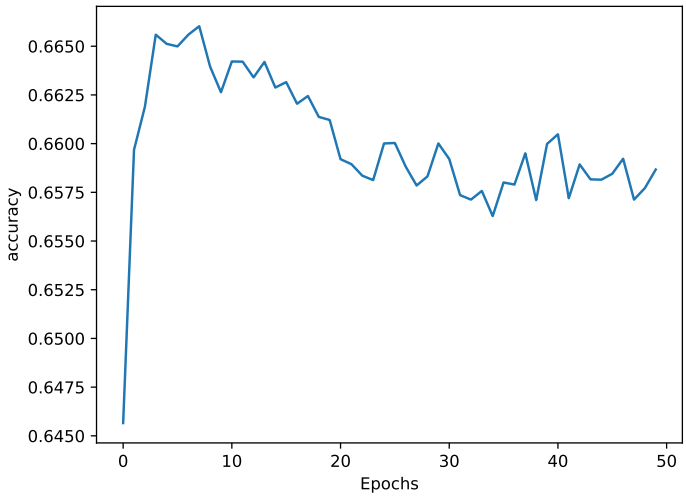

Figure 2.5: Training Accuracy Plot on BiLSTM Model

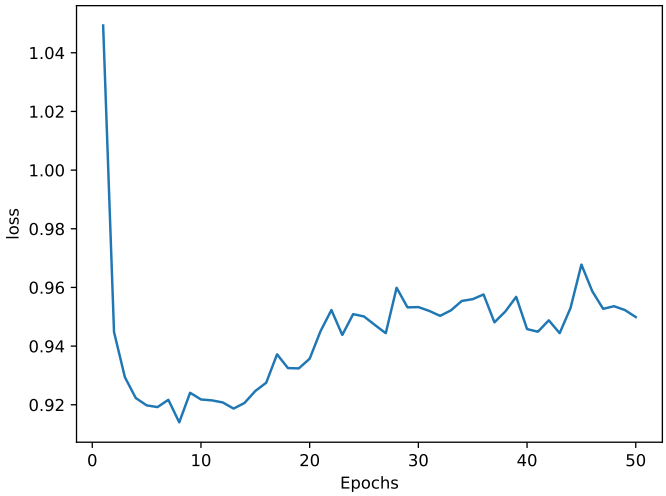

Figure 2.6: Training Loss Plot on BiLSTM Model

References

[1] M. Berry, "Welcome to scratch 3!," 2019.

[2] K. Heafield, *kenlm*, https://github.com/kpu/kenlm, 2013.

[3] E. Loper and S. Bird, "Nltk: The natural language toolkit," *arXiv preprint cs/0205028*, 2002.

[4] B. Jang, M. Kim, G. Harerimana, S.-u. Kang, and J. W. Kim, "Bi-lstm model to increase accuracy in text classification: Combining word2vec cnn and attention mechanism," *Applied Sciences*, vol. 10, no. 17, p. 5841, 2020.

[5] J. Klump, L. Wyborn, M. Wu, J. Martin, R. R. Downs, and A. Asmi, "Versioning data is about more than revisions: A conceptual framework and proposed principles," *Data Science Journal*, vol. 20, no. 1, p. 12, 2021.

[6] S. Chacon, *Git SCM*, https://git-scm.com, Accessed: 2023-11-12, 2005.

[7] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "The Promises and Perils of Mining GitHub," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 92–101.

[8] S. Iwuchukwu, *ScratchParser*, https://github.com/UncleSamTech/thesis_code/blob/main/parser/scratch_parser.py, Accessed: 2024-05-28, 2023.

[9] L. Grunske, K. Winter, and N. Yatapanage, "Defining the abstract syntax of visual languages with advanced graph grammars—a case study based on behavior trees," *Journal of Visual Languages & Computing*, vol. 19, no. 3, pp. 343–379, 2008.

[10] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th international conference on World Wide Web*, 2016, pp. 263–273.

[11] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. Devanbu, "On the naturalness of software," *Communications of the ACM*, vol. 59, no. 5, pp. 122–131, 2016.

[12] O. Kramer and O. Kramer, "Scikit-learn," *Machine learning for evolution strategies*, pp. 45–53, 2016.

[13] nltkproj, *nltk*, https://www.nltk.org/api/nltk.tokenize.word_tokenize.html, 2023.

[14] nltkproj, *nltk$_p$add*, https://www.nltk.org/api/nltk.lm.preprocessing.html, 2023.

[15] nltkproj, *nltkmle*, https://www.nltk.org/api/nltk.lm.html, 2023.

[16] D. H. Klatt, "Review of the arpa speech understanding project," *The Journal of the Acoustical Society of America*, vol. 62, no. 6, pp. 1345–1366, 1977.

[17] K. Heafield, I. Pouzyrevsky, J. H. Clark, and P. Koehn, "Scalable modified kneser-ney language model estimation," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2013, pp. 690–696.

[19] TensorProj, *Tensorflow pad sequences*, https://www.tensorflow.org/api_docs/python/tf/keras, 2023.

[0] G. Van Rossum and F. L. Drake Jr, *Python tutorial*, 1995.

[0] E. A. Gehan, "A generalized wilcoxon test for comparing arbitrarily singly-censored samples," *Biometrika*, vol. 52, no. 1-2, pp. 203–224, 1965.

[0] D. R. Anderson, K. P. Burnham, and W. L. Thompson, "Null hypothesis testing: Problems, prevalence, and an alternative," *The journal of wildlife management*, pp. 912–923, 2000.

[0] R. Paine, "What is gossip about? an alternative hypothesis," *Man*, vol. 2, no. 2, pp. 278–285, 1967.

[0] R. A. Thisted, "What is a p-value," *Departments of Statistics and Health Studies*, 1998.