Playing Hangman with Large Language Models

Manurag Khullar, Vaibhav Sahu
TA Mentor: Yunshuang Li

May 2024

1 Abstract

Hangman is a classic word-guessing game where one player thinks of a word and the other tries to guess it by suggesting letters within a certain number of guesses. In this project, we have trained Google's CANINE transformer to play Hangman, aiming to teach the model strategic letter guessing based on partial word clues. Our approach involved modeling the game's mechanics and states, allowing the AI to navigate various scenarios effectively. The results were promising, with the model achieving 63% success in guessing words correctly within the standard six tries and further improvement of accuracy to 86% was noted when additonal tries were allowed. This demonstrates the potential of using sophisticated language models in simple, challenging cognitive tasks.

2 Introduction

Hangman's objective is to correctly guess a hidden word by suggesting letters within a limited number of attempts. Despite its apparent simplicity, Hangman presents a significant cognitive challenge, requiring strategic guessing based on partial information. This project aims to leverage modern natural language processing (NLP) techniques, specifically Google's CANINE transformer model, to develop an AI capable of playing Hangman. The importance of this problem lies in its broader implications for NLP and machine learning. Developing an AI that can effectively play Hangman showcases the potential of tokenization-free models in handling character-level text, which is essential for languages and applications where traditional tokenization methods fall short. Additionally, this project highlights the versatility of advanced language models in adapting to various tasks, including games that involve strategic decision-making and pattern recognition.

In our proposed model, the primary inputs and outputs are defined as follows:

- **Inputs**
  - **Game state** (s_t): A representation of the current state of the word, showing correct guesses and placeholders for remaining letters. The game state also includes guess letters.
- **Outputs**
  - **Next letter guess** (L_t): The predicted letter the model selects as the next guess.
  - **Updated game state** (s_{t+1}): The game state after incorporating the new guess, reflecting any correct guesses and remaining placeholders.

3 Background

Several AI models for Hangman have been developed, each with its approach and limitations.

1. We used intuition inspired by Microsoft Azure's training for Hangman players, which can be found here[1]. It helped us with modeling the ML task. However, their project also gave us some good insights into improving our model performance. We plan to enhance our performance by further fine-tuning our model.
   Limitation: Their approach is nice; however, they try to train the model from self-play from the start. This is not optimal as the model takes more time to learn, initially being a weak player.

2. B. Clark, Jonathan H., et al., ”Canine: Pre-training an Efficient Tokenization- Free Encoder for Language Representation.” [3] This work on the CANINE-S transformer model provides a foundation for understanding how transformers can be adapted for character-level text processing and will serve as a technical reference for our model development.
   Limitation: This is more about generalized Language modeling. We need to tweak it before using it in our Hangman game player.

4 Summary of Our Contributions

We have thought of a pre-training task that would work great with a character-level LLM. We can effectively divide our project into the following tasks and contributions:

1. Analyzing the complexity of the problem and conducting an exploratory analysis to identify effective strategies. Ultimately, we aim to maintain an end-to-end, strictly ML-based approach. However, identifying strategies will aid in debugging our model's performance and assessing its ability to learn and replicate them.

2. Pre-training the model: Our plan involves simulating hangman games in a Monte Carlo-based fashion and encoding them into text to feed into our language model. This serves as the pre-training task we have devised.

3. Self-play fine-tuning: Our model benefits greatly from learning to play itself when it has figured out the initial distribution. We choose a smaller dataset to train the model for this. It differs from Microsoft's approach of training through self-play from the start.

4. Lastly, we aim to enhance our model's performance and analyze its errors and shortcomings. This analysis will enable us to derive insights from the project, outline its contributions, and determine future avenues for improvement, if necessary.

5 Detailed Description of Contributions

1. **Set up the problem**: Hangman can be considered a Markovian Process. Given this conditional time-series-like dependency, we thought sequence modeling could be done using RNNs and Transformers. Effectively, we model the problem as follows:

   Our actions are decisions that our model takes, given the state at a given time step. Hangman states can be completely defined by keeping track of all the guessed letters and the state of our word. For example, consider the example of “birthday.” Say we guessed the letters “e,” “i,” “a,” and “s.” The state of our word becomes “_ i _ _ _ _ a _.” Optimally, our model has to learn to condition its guess based on the updated state at each time step. Notice that the action is a letter guess. We only have 26 letters, so our action can be modeled as a classification task that predicts the correct letter or the "class" out of 26 letters ("classes").

2. **Proposed state**: We can text-encode all the states. Essentially, we could feature design this and pass encoded vectors for the letters that have already been guessed and traverse the game state as a sequence of one-hot vectors denoting letters. However, we know transformers and LLMs are great at learning from text-encoded information. To ensure our transformer learns this mapping, we will discuss some ways to ensure this. For now, the state information is stored in the following way:

   ```
   [CLS] <GUESSED_LETTERS> [SEP] <HANGMAN_STATE> [SEP]
   ```

   The `HANGMAN_STATE` is itself encoded neatly. For example, the hidden letters missing in our word are modeled using `[MASK]` tokens. In this way, our problem is ready to be used as a pre-training task for our Language Model.

   ```
   [CLS] eias [SEP] [MASK] i [MASK] [MASK] [MASK] [MASK] a [MASK] [SEP]
   ```

3. **Self-play finetuning** As the name suggests, we make the model learn from its mistakes. It plays the game 100k times on different words in 10 iterations. For each iteration, we sample 10k words, and the model iteratively corrects itself through gradient-descent

5.1 Method

Machine Learning Task: The objective of our agent is to minimize the number of tries it uses. However, how do we model the loss? There are 2 ways to approach this problem.

A. Multilabel classification task: We can model all the disguised letters that have not been guessed as the correct labels. The label would then be just the encoded vector with 1's on all the correct letters that have not been guessed. This gives the model more slack to work with. It can work well if we train the model for longer. However, given our limited resources, I realized that the more restrictive approach had a smaller search space and gave results faster.

B. Multiclass classification task: This is more restrictive. We predict the most frequent missing letter. The probability of letters can be modeled based on their frequency in our word. We keep track of the frequencies and then normalize them to get the probabilities. This 26-dimensional vector is then used as the ground truth for our model. This way, it learns to predict the most likely letter.

We used the Multiclass classification task as this is more restrictive, has a smaller search space, and needs less training. The loss function is simply the cross-entropy loss between this normalized probability vector and the softmax output from our model,

$$f(s_t, i) = \frac{\sum_j \delta_j \cdot I_{x_j = x_i}}{\sum_i \sum_j \delta_j \cdot I_{x_j = x_i}}$$

where x_i is the i-th letter of the alphabet, and x_j is the letter at the j-th position of the word if it has not been guessed already. The loss function is given by:

$$L(f(s_t), o_{LM}) = \sum_i -f(s_t, i) \log(o_{LM}(i))$$

where o_{LM} is the 26-dimensional softmax output of our classification model. Our model effectively learns to model the conditional distribution of letters.

Model Selection: It's time to select the best model for our ML task. We need a large enough model to learn this end-to-end mapping we have designed. However, it is interesting to note that the token-level structure in our inputs is characters. Therefore, we chose a character-level model. As we stated earlier, we also found a great model called CANINE. Google's CANINE model can learn language-level concepts even though it is a character-level model. It is motivated to make it tokenization agnostic. However, we benefit from its character-level featurization, and the scale is not too huge to pre-train it from scratch (121 Million parameters).

5.2 Playing Hangman with CANINE Algorithm

1. **State Representation**
   - Define the game state s_t at time step t as we described earlier.
2. **Pre-training**
   - Generate training data through simulated gameplay using biased sampling strategies to cover a wide array of game states.
   - Train the transformer model using these game states:
     - Input: Encoded game state s_t.
     - Output: Next letter to guess.
     - Loss function: Cross-entropy loss between the model's predictions and the actual next letters.
3. **Self-play Fine-tuning**
   - Utilize the pre-trained model to play against itself, refining its predictions based on actual game outcomes.
   - Adjust the model's parameters to minimize the number of incorrect guesses, thus optimizing the prediction accuracy.
   - Continue the iterative process of self-play to enhance strategic decision-making under varying game states.
4. **Evaluation**
   - Measure the model's performance through metrics such as accuracy of letter prediction and overall game-winning rates.
   - Analyze the effectiveness of the training by testing the model on unseen words and comparing the success rate against baseline models.

5.3 Experiments and Results

5.3.1 Exploratory Data Analysis

On the entire dataset, we analyze the distribution of words by their length and also the natural calling order of the dataset. It resembles a Poisson distribution with a mode length of about 9 letters. Here is the distribution of word length:

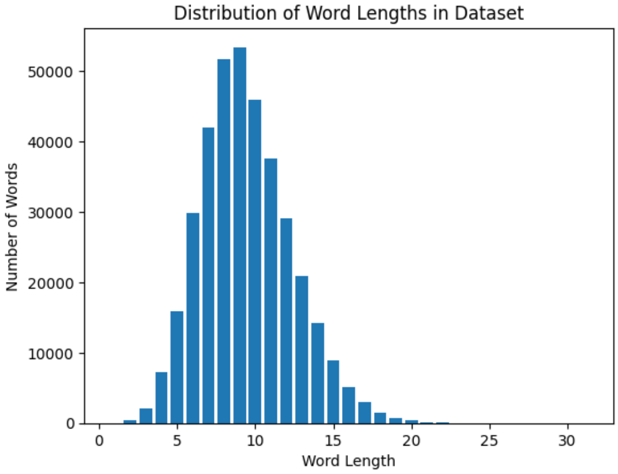
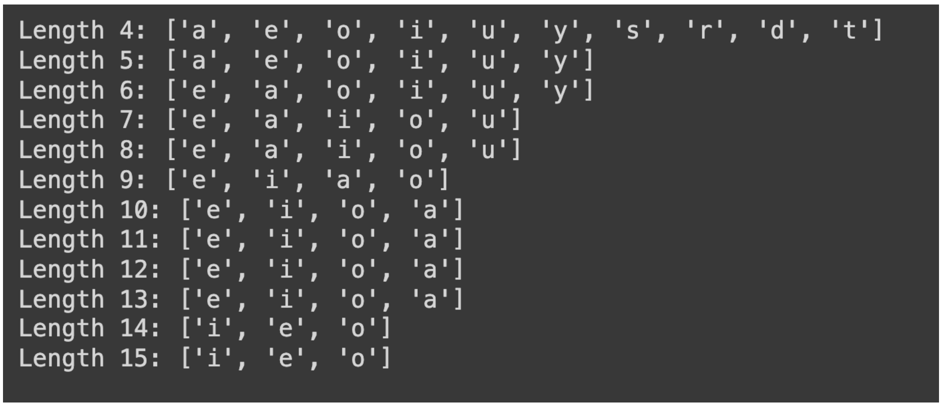


Figure 1: Word length distribution

The calling order [2] is the order of letters to call without any correct guesses in Hangman. We can obtain the natural calling order by conditionally calculating the most probable letter and removing words containing that letter. Iteratively doing this, we can obtain the final calling order.



```
Length 4: ['a', 'e', 'o', 'i', 'u', 'y', 's', 'r', 'd', 't']
Length 5: ['a', 'e', 'o', 'i', 'u', 'y']
Length 6: ['e', 'a', 'o', 'i', 'u', 'y']
Length 7: ['e', 'a', 'i', 'o', 'u']
Length 8: ['e', 'a', 'i', 'o', 'u']
Length 9: ['e', 'i', 'a', 'o']
Length 10: ['e', 'i', 'o', 'a']
Length 11: ['e', 'i', 'o', 'a']
Length 12: ['e', 'i', 'o', 'a']
Length 13: ['e', 'i', 'o', 'a']
Length 14: ['i', 'e', 'o']
Length 15: ['i', 'e', 'o']
```

Figure 2: Calling Order of our Dataset

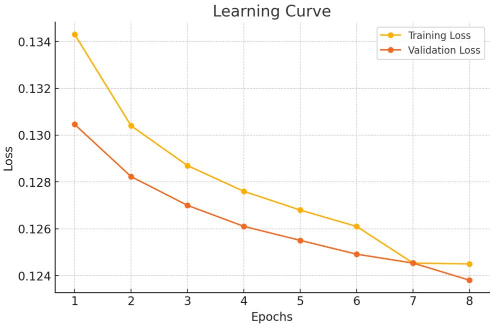
Our dataset has 400k words. For each testing and validation, we take 10k words and use the rest to train our model.

5.4 Baseline Model

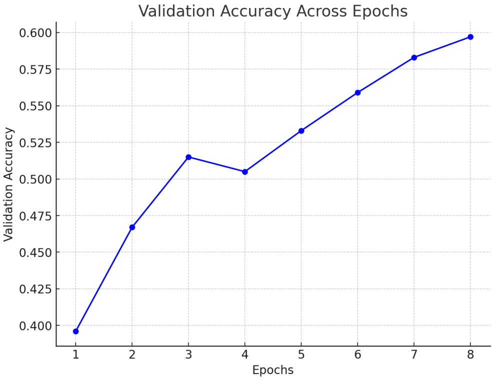
We created a baseline model to play Hangman by predicting letters using n-gram models (unigrams, bigrams, and trigrams). Model reads a list of words from a file, builds the n-gram models from these words, and uses these models to guess letters in a simulated Hangman game. The ngram guesser function calculates the probabilities for each character based on the current game state and previously guessed letters, selecting the character with the highest probability that hasn't been guessed. Then we simulate the game, updating the mask of the word and counting mistakes until the word is guessed or the maximum number of mistakes is reached. We test the accuracy on same datasets which turns out to be 23% for getting correct word in 6 tries, and 56% for 10 tries.

5.5 Pre-training

The pre-training task uses 380k words to train the model. We were able to achieve 59% validation accuracy. Here is the learning curve and the accuracy plot of training our model on the task:



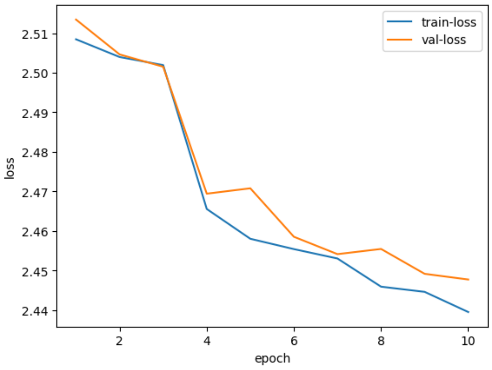
(a) Learning Curve



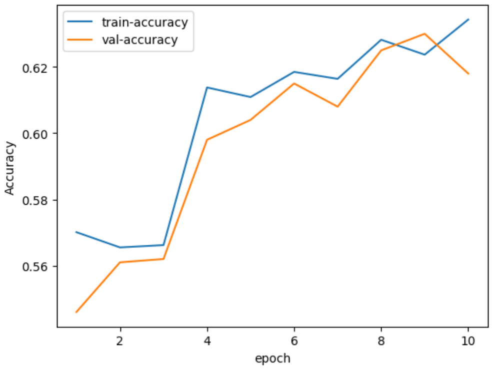
(b) Validation accuracy plot for pre-training

5.6 Self-play Finetuning

For self-play fine-tuning, we got a validation accuracy of 63.5%! The model could be tuned further with better hyperparameters. However, even with the current model settings, we also get an accuracy of 63% on our test set.



(a) Learning Curve



(b) Accuracy plot for self-play

To gain further insight into a model performance, we also plot the accuracy of our model with word length:

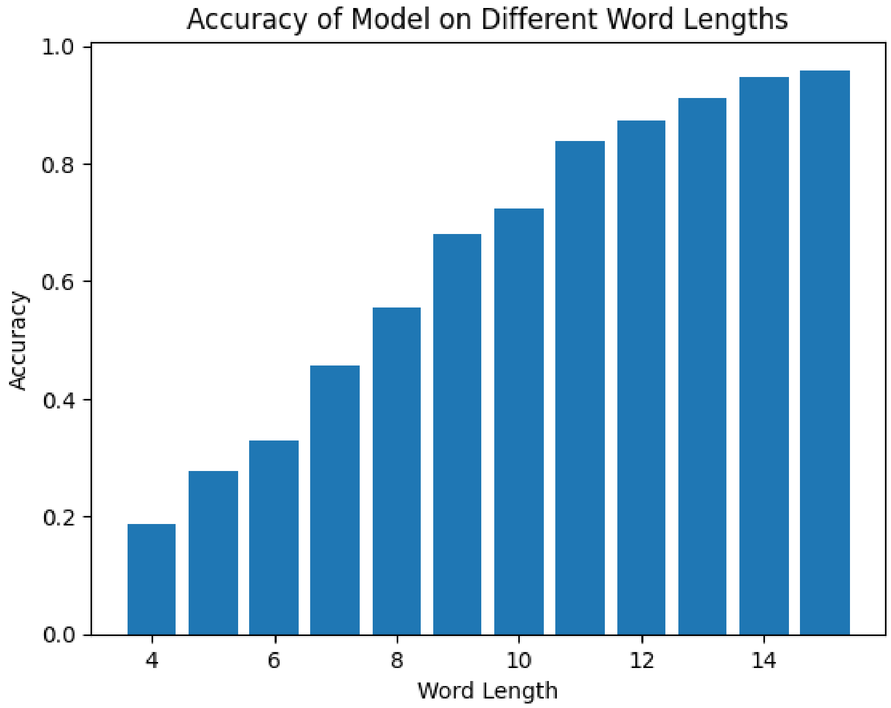


Figure 5: Accuracy vs word length

	Baseline N-gram Model	Modified CANINE Model
6-guesses	23%	63%
10-guesses	56%	86%

Table 1: Test accuracy comparison between Baseline N-gram Model and Modified CANINE Model

6 Compute and other resources used

We used high-performance computing resources with GPUs to train our model using deep learning algorithms. Given the size of our model and dataset, pre-training was expensive and took 12 hours on an A100 GPU.

7 Conclusion

In conclusion, this project has successfully demonstrated the capability of large language models, specifically Google's CANINE transformer, to tackle the cognitive challenge of playing Hangman. By formulating the game as a sequence modeling task and leveraging techniques like pre-training on simulated games and fine-tuning self-play, our model achieved promising results, correctly guessing 63% of words within the standard number of tries. This showcases the potential of tokenization-free models in handling character-level text and highlights the versatility of advanced language models in adapting to tasks involving strategic decision-making and pattern recognition. While there is room for further improvement, such as through hyperparameter tuning and exploring alternative modeling approaches, this project serves as a compelling demonstration of the capabilities of modern natural language processing in tackling seemingly simple yet cognitively complex problems.

References

[1] Microsoft Azure. Hangman. `https://github.com/Azure/Hangman/tree/master`, 2024. Accessed: 2024-05-15.

[2] Nick Berry. Data genetics, 2012. Accessed: 2023-05-15.

[3] Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *arXiv preprint arXiv:2103.06874*, 2021. TACL Final Version.

8 Appendix

8.1 Broader Dissemination Information:

Your report title and the list of team members will be published on the class website. Would you also like your pdf report to be published? YES, the report could be published.

If your answer to the above question is yes, are there any other links to github / youtube / blog post / project website that you would like to publish alongside the report? If so, list them here.

Yes, you can use the following github link:
https://github.com/vsa1920/Hangman-with-Transformers

8.2 Timeline

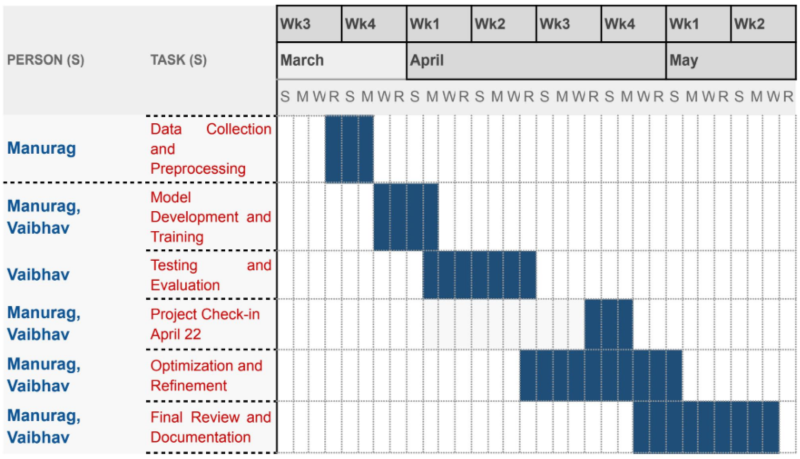
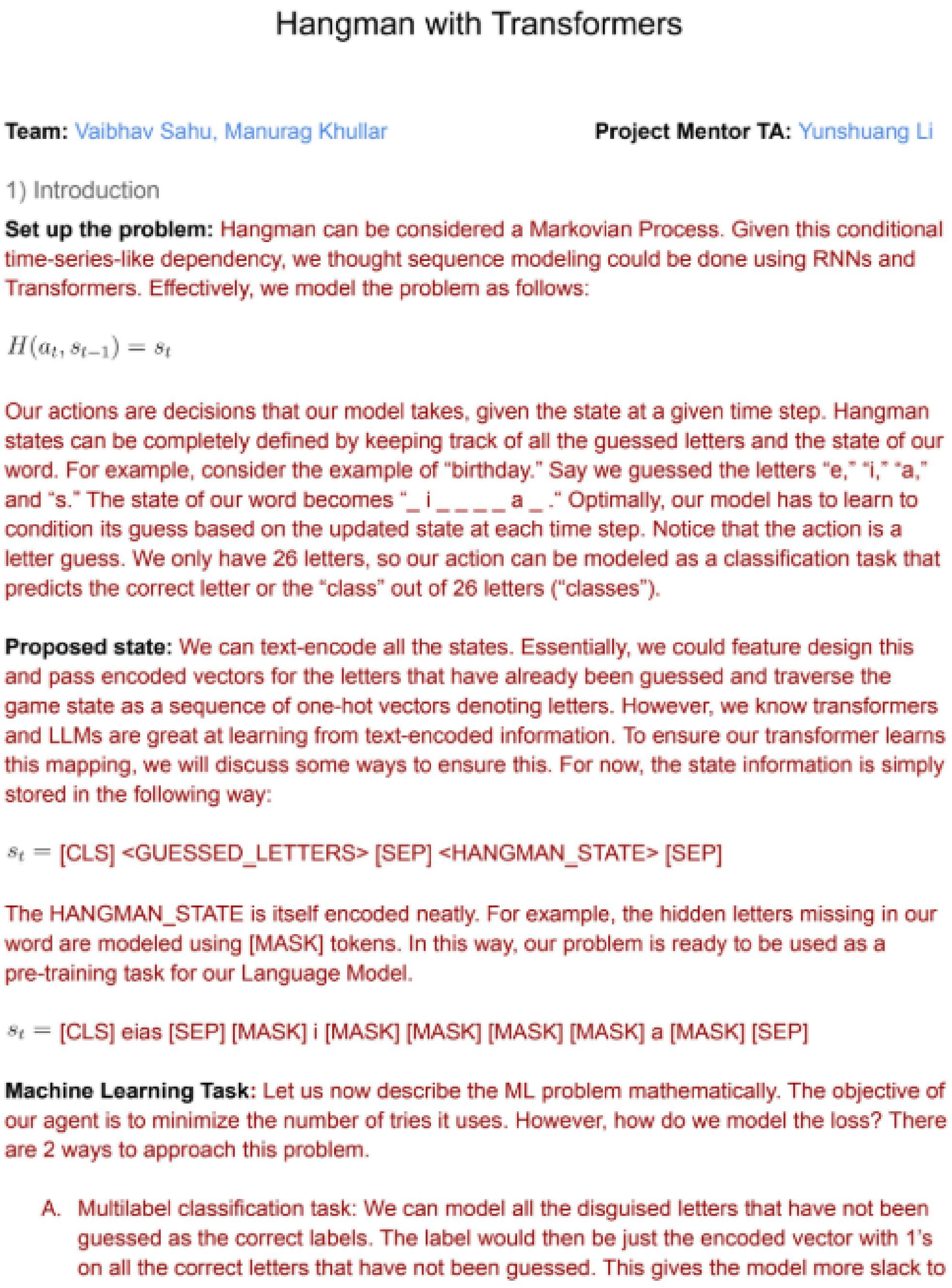


Figure 6: Timeline

8.3 Project Check-in



Hangman with Transformers

Team: Vaibhav Sahu, Manurag Khullar **Project Mentor TA:** Yunshuang Li

1) Introduction

Set up the problem: Hangman can be considered a Markovian Process. Given this conditional time-series-like dependency, we thought sequence modeling could be done using RNNs and Transformers. Effectively, we model the problem as follows:

$H(a_t, s_{t-1}) = s_t$

Our actions are decisions that our model takes, given the state at a given time step. Hangman states can be completely defined by keeping track of all the guessed letters and the state of our word. For example, consider the example of "birthday." Say we guessed the letters "e," "i," "a," and "s." The state of our word becomes "_ i _ _ _ _ a _ ." Optimally, our model has to learn to condition its guess based on the updated state at each time step. Notice that the action is a letter guess. We only have 26 letters, so our action can be modeled as a classification task that predicts the correct letter or the "class" out of 26 letters ("classes").

Proposed state: We can text-encode all the states. Essentially, we could feature design this and pass encoded vectors for the letters that have already been guessed and traverse the game state as a sequence of one-hot vectors denoting letters. However, we know transformers and LLMs are great at learning from text-encoded information. To ensure our transformer learns this mapping, we will discuss some ways to ensure this. For now, the state information is simply stored in the following way:

s_t = [CLS] <GUESSED_LETTERS> [SEP] <HANGMAN_STATE> [SEP]

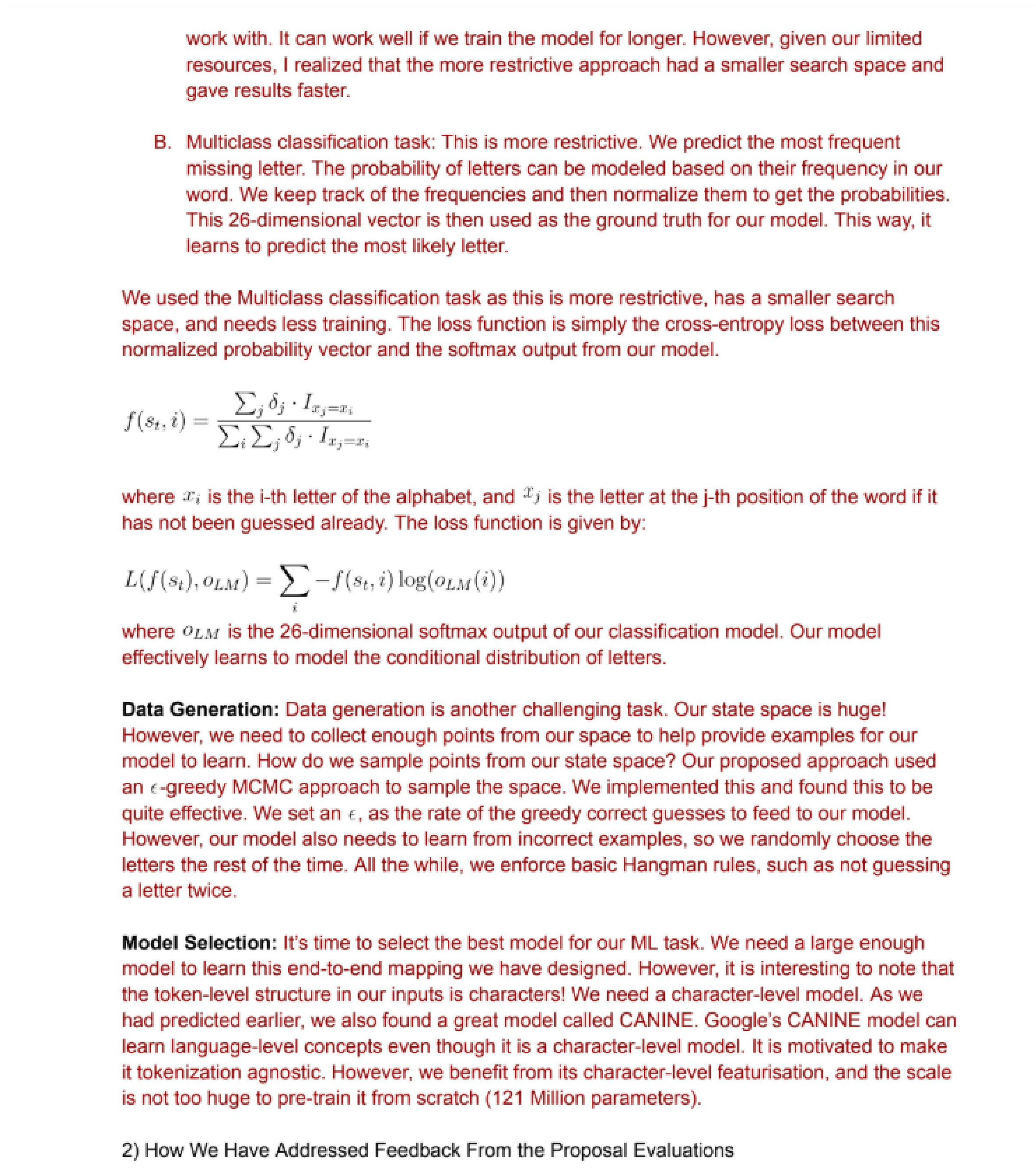
The HANGMAN_STATE is itself encoded neatly. For example, the hidden letters missing in our word are modeled using [MASK] tokens. In this way, our problem is ready to be used as a pre-training task for our Language Model.

s_t = [CLS] eias [SEP] [MASK] i [MASK] [MASK] [MASK] [MASK] a [MASK] [SEP]

Machine Learning Task: Let us now describe the ML problem mathematically. The objective of our agent is to minimize the number of tries it uses. However, how do we model the loss? There are 2 ways to approach this problem.

A. Multilabel classification task: We can model all the disguised letters that have not been guessed as the correct labels. The label would then be just the encoded vector with 1's on all the correct letters that have not been guessed. This gives the model more slack to

Figure 7: Project Check-in Screenshot1



work with. It can work well if we train the model for longer. However, given our limited resources, I realized that the more restrictive approach had a smaller search space and gave results faster.

B. Multiclass classification task: This is more restrictive. We predict the most frequent missing letter. The probability of letters can be modeled based on their frequency in our word. We keep track of the frequencies and then normalize them to get the probabilities. This 26-dimensional vector is then used as the ground truth for our model. This way, it learns to predict the most likely letter.

We used the Multiclass classification task as this is more restrictive, has a smaller search space, and needs less training. The loss function is simply the cross-entropy loss between this normalized probability vector and the softmax output from our model.

$$f(s_t, i) = \frac{\sum_j \delta_j \cdot I_{x_j = x_i}}{\sum_i \sum_j \delta_j \cdot I_{x_j = x_i}}$$

where x_i is the i-th letter of the alphabet, and x_j is the letter at the j-th position of the word if it has not been guessed already. The loss function is given by:

$$L(f(s_t), o_{LM}) = \sum_i -f(s_t, i) \log(o_{LM}(i))$$

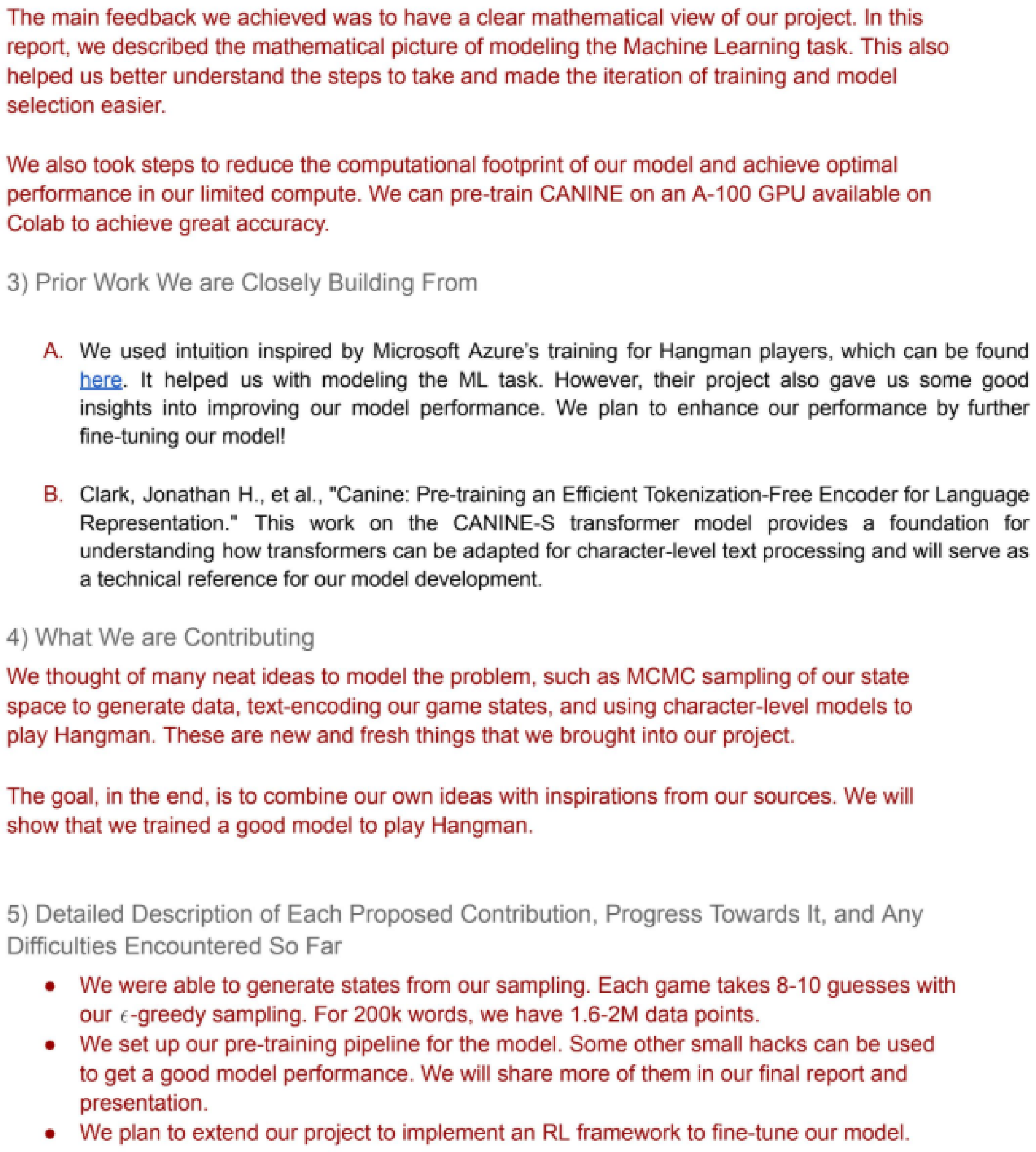
where o_{LM} is the 26-dimensional softmax output of our classification model. Our model effectively learns to model the conditional distribution of letters.

Data Generation: Data generation is another challenging task. Our state space is huge! However, we need to collect enough points from our space to help provide examples for our model to learn. How do we sample points from our state space? Our proposed approach used an ϵ-greedy MCMC approach to sample the space. We implemented this and found this to be quite effective. We set an ϵ, as the rate of the greedy correct guesses to feed to our model. However, our model also needs to learn from incorrect examples, so we randomly choose the letters the rest of the time. All the while, we enforce basic Hangman rules, such as not guessing a letter twice.

Model Selection: It's time to select the best model for our ML task. We need a large enough model to learn this end-to-end mapping we have designed. However, it is interesting to note that the token-level structure in our inputs is characters! We need a character-level model. As we had predicted earlier, we also found a great model called CANINE. Google's CANINE model can learn language-level concepts even though it is a character-level model. It is motivated to make it tokenization agnostic. However, we benefit from its character-level featurisation, and the scale is not too huge to pre-train it from scratch (121 Million parameters).

2) How We Have Addressed Feedback From the Proposal Evaluations

Figure 8: Project Check-in Screenshot2



The main feedback we achieved was to have a clear mathematical view of our project. In this report, we described the mathematical picture of modeling the Machine Learning task. This also helped us better understand the steps to take and made the iteration of training and model selection easier.

We also took steps to reduce the computational footprint of our model and achieve optimal performance in our limited compute. We can pre-train CANINE on an A-100 GPU available on Colab to achieve great accuracy.

3) Prior Work We are Closely Building From

A. We used intuition inspired by Microsoft Azure's training for Hangman players, which can be found here. It helped us with modeling the ML task. However, their project also gave us some good insights into improving our model performance. We plan to enhance our performance by further fine-tuning our model!

B. Clark, Jonathan H., et al., "Canine: Pre-training an Efficient Tokenization-Free Encoder for Language Representation." This work on the CANINE-S transformer model provides a foundation for understanding how transformers can be adapted for character-level text processing and will serve as a technical reference for our model development.

4) What We are Contributing

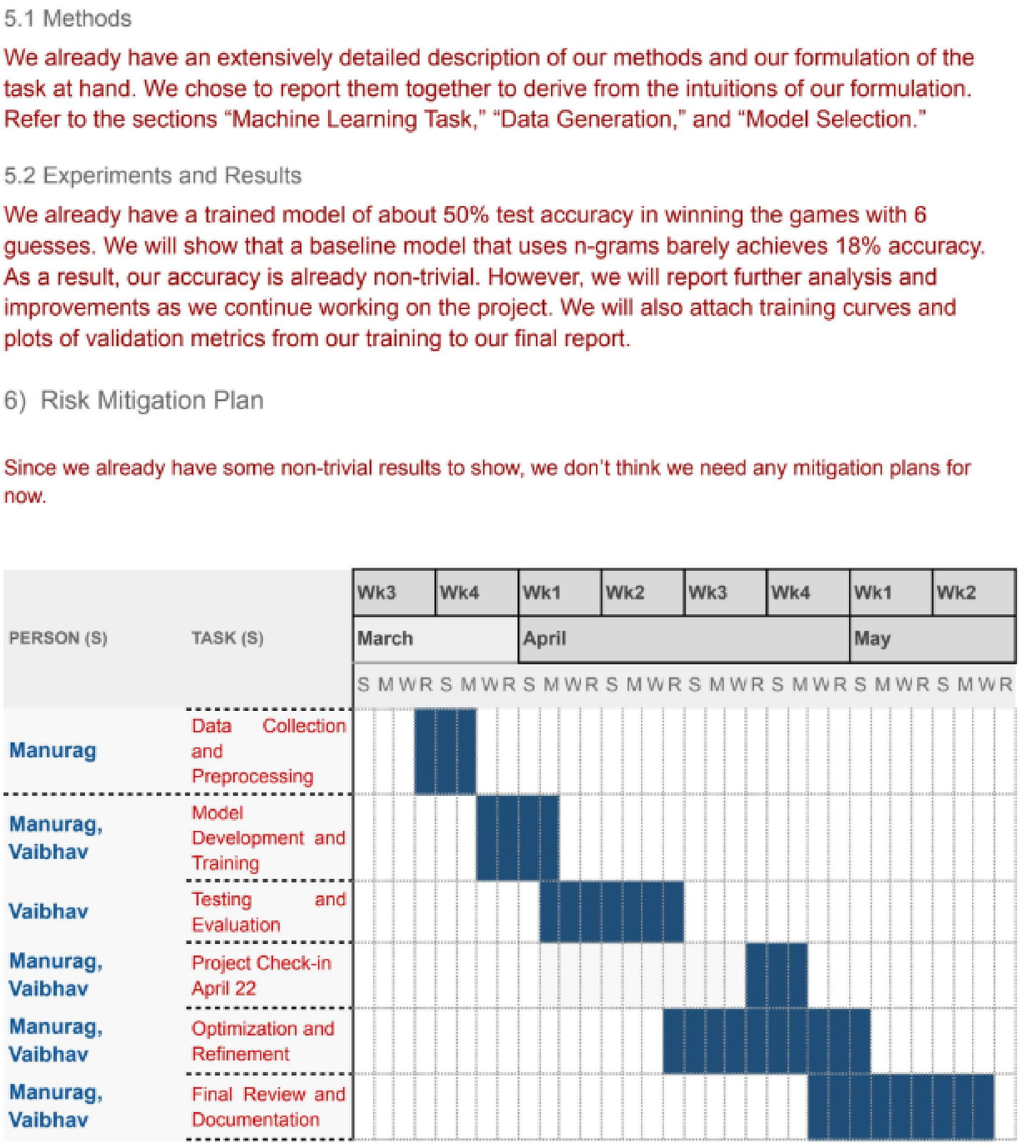
We thought of many neat ideas to model the problem, such as MCMC sampling of our state space to generate data, text-encoding our game states, and using character-level models to play Hangman. These are new and fresh things that we brought into our project.

The goal, in the end, is to combine our own ideas with inspirations from our sources. We will show that we trained a good model to play Hangman.

5) Detailed Description of Each Proposed Contribution, Progress Towards It, and Any Difficulties Encountered So Far

- We were able to generate states from our sampling. Each game takes 8-10 guesses with our ϵ-greedy sampling. For 200k words, we have 1.6-2M data points.
- We set up our pre-training pipeline for the model. Some other small hacks can be used to get a good model performance. We will share more of them in our final report and presentation.
- We plan to extend our project to implement an RL framework to fine-tune our model.

Figure 9: Project Check-in Screenshot3



5.1 Methods

We already have an extensively detailed description of our methods and our formulation of the task at hand. We chose to report them together to derive from the intuitions of our formulation. Refer to the sections “Machine Learning Task,” “Data Generation,” and “Model Selection.”

5.2 Experiments and Results

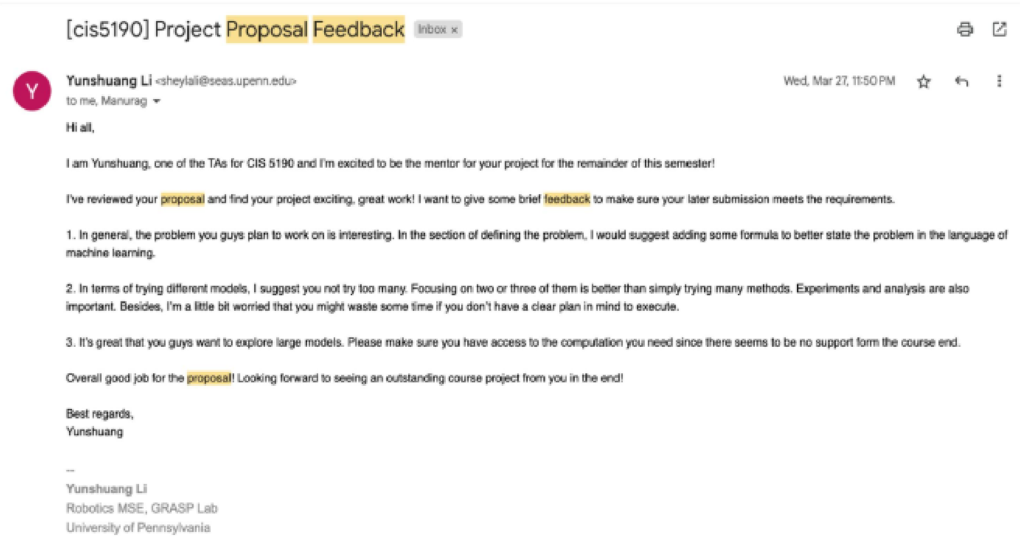
We already have a trained model of about 50% test accuracy in winning the games with 6 guesses. We will show that a baseline model that uses n-grams barely achieves 18% accuracy. As a result, our accuracy is already non-trivial. However, we will report further analysis and improvements as we continue working on the project. We will also attach training curves and plots of validation metrics from our training to our final report.

6) Risk Mitigation Plan

Since we already have some non-trivial results to show, we don’t think we need any mitigation plans for now.

PERSON (S)	TASK (S)	Wk3				Wk4				Wk1				Wk2				Wk3				Wk4				Wk1				Wk2			
		March								April																May							
		S	M	W	R	S	M	W	R	S	M	W	R	S	M	W	R	S	M	W	R	S	M	W	R	S	M	W	R	S	M	W	R
Manurag	Data Collection and Preprocessing				■	■	■																										
Manurag, Vaibhav	Model Development and Training							■	■	■	■																						
Vaibhav	Testing and Evaluation										■	■	■	■	■	■	■																
Manurag, Vaibhav	Project Check-in April 22																				■	■	■										
Manurag, Vaibhav	Optimization and Refinement																■	■	■	■	■	■	■	■	■	■							
Manurag, Vaibhav	Final Review and Documentation																							■	■	■	■	■	■	■	■	■	

Figure 10: Project Check-in Screenshot4



[cis5190] Project Proposal Feedback

Yunshuang Li <sheylali@seas.upenn.edu>
to me, Manurag

Wed, Mar 27, 11:50 PM

Hi all,

I am Yunshuang, one of the TAs for CIS 5190 and I'm excited to be the mentor for your project for the remainder of this semester!

I've reviewed your proposal and find your project exciting, great work! I want to give some brief feedback to make sure your later submission meets the requirements.

1. In general, the problem you guys plan to work on is interesting. In the section of defining the problem, I would suggest adding some formula to better state the problem in the language of machine learning.

2. In terms of trying different models, I suggest you not try too many. Focusing on two or three of them is better than simply trying many methods. Experiments and analysis are also important. Besides, I'm a little bit worried that you might waste some time if you don't have a clear plan in mind to execute.

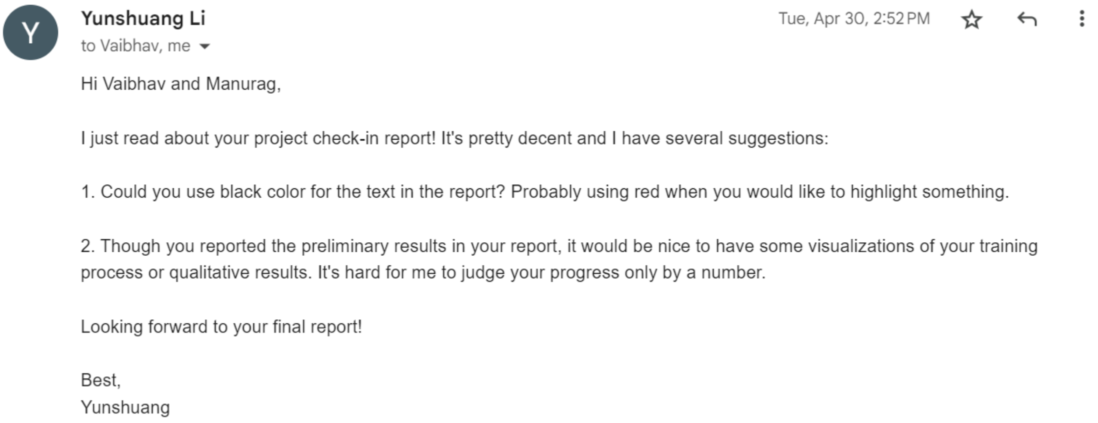
3. It's great that you guys want to explore large models. Please make sure you have access to the computation you need since there seems to be no support form the course end.

Overall good job for the proposal! Looking forward to seeing an outstanding course project from you in the end!

Best regards,
Yunshuang

--
Yunshuang Li
Robotics MSE, GRASP Lab
University of Pennsylvania

Figure 11: Project Check-in Screenshot5



Yunshuang Li
to Vaibhav, me

Tue, Apr 30, 2:52 PM

Hi Vaibhav and Manurag,

I just read about your project check-in report! It's pretty decent and I have several suggestions:

1. Could you use black color for the text in the report? Probably using red when you would like to highlight something.

2. Though you reported the preliminary results in your report, it would be nice to have some visualizations of your training process or qualitative results. It's hard for me to judge your progress only by a number.

Looking forward to your final report!

Best,
Yunshuang

Figure 12: Project Check-in Screenshot6