ChePT-2: Advancing the Application of Deep Neural Transformer Models to Chess Move Prediction and Self-Commentary

Henry Mellsop
Department of Computer Science
Stanford University
hmellsop@stanford.edu

Collin Schlager
Department of Computer Science
Stanford University
schlager@stanford.edu

Colton Swingle
Department of Computer Science
Stanford University
cswingle@stanford.edu

1 Introduction

For many years, Chess has been the subject of great interest in Computer Science and Artificial Intelligence. Only half a decade after Chess Grandmaster and World Champion Garry Kasparov's 1985 claim that no computer program would ever best him, IBM's Deep Blue reached a proficiency significantly greater than any human on Earth [1]. Since then, 'Chess Engines' - computer programs designed to excel at playing Chess - have improved dramatically. New techniques and approaches, particularly within the domain of reinforcement learning, have redefined how programmers approach the game [2].

Modern state-of-the-art Chess Engines are all stateful; they observe a single board configuration and rely on a combination of massive depth-limited searches, human tuned weightings of configurations and pieces, Monte-Carlo Tree Searches, and neural-network based evaluation functions to achieve excellent inference [3]. These engines all subscribe to the Markov Property - observing a static board state with no knowledge of what was played beforehand can fully define define the ideal action to take. Moreover, none of these models provide any explicit form of insight into their own decision making. Human observers - even Chess experts and Grandmasters - are often left baffled by the decisions that the engines make. As such, developing a Chess engine that can explain its own decision-making is a fascinating area of inquiry.

Inspired by 'blindfold chess' - a form of Chess where players are unable to observe the board, and instead play games by listening to their opponent's move, before speaking their own - we forgoe this traditional stateful representation of chess, and instead reframe Chess as a sequence modelling problem. A game of Chess can be recast as a language, where each word is a move, and the commentary could be included directly in this new language. Essentially, we sought to build a 'blindfolded chess' model that plays just as a blind-folded human might. Our goal was to combine the (related) tasks of both playing Chess and performing self-commentary. Such a model would be fantastically useful to Chess students, but also would present a number of fascinating questions about model explainability, and re-casting traditional reinforcement learning tasks as sequence modelling tasks.

Formally, our model takes as input a string of Chess moves, and returns the 'ideal' next move according to its own analysis. It then also optionally returns a short English commentary of the move that it generated. As such, you can play this model against humans, against other Engines, or against itself. Further, you can prompt the model to comment on its latest move.

2 Related Work

In the space of applying generative sequence modelling to Chess, the literature is quite limited. Noever et al. [4] were the first to explore applying this class of model - specifically a pretrained GPT-2 model - to Chess in 2020. Their approach of re-casting Chess as a sequential language task forms a key basis of inspiration for our approach. Their model used character-level embeddings to excel solely at the task of playing Chess, and we have included their relevant gameplay statistics in comparison to ours in the experiment section. They also present a number of interesting insights with respect to training strategies, such as mixing black and white-win games, and filtering training data based off of ELO in order to imbue the model with a knowledge of not only basic Chess rules, but also more advanced strategy.

With respect to Chess commentary, the literature is somewhat more standardised. Data scraped from the `gameknot.com` website appears to be standard in the literature [5] [6]. These papers all use human reviewers to analyse predicted commentary fluency, accuracy, insights and overall interpretability. They also all use BLEU scores from natural language literature, but a number of different papers cite notable difficulties with such automated metrics in the relatively data-sparse and unconstrained output space that is commentary [7] [5] [6]. As such, (especially given the space constraints of the write-up) we chose to forgoe automated commentary evaluation metrics, instead focusing on automating the analysis of our Chess-playing objective, and analysing commentary by hand. Importantly, no models we have found in the literature perform both move prediction and commentary.

Earlier this year, we applied deep end-to-end transformer neural models to this move prediction and self-commentary task [7]. The model we developed - ChePT-1 - was relatively successful in this space, but suffered a number of drawbacks. Specifically, the model struggled with English grammar and spelling when predicting commentary, owing in part to the lack of English-language pretraining objectives. Moreover, the model suffered significantly in mid-to-late game when predicting Chess moves, in part due to the character-level embedding strategy that we employed. As such, our goal for this project was to build significantly on these findings by completely re-imagining our training objectives, model architecture, and embedding strategies.

3 Datasets

3.1 Chess Datasets

Given that we are not providing the model with an explicit set of Chess rules, we require a large corpus of training data. We obtained the Kaggle `Milesh1` Chess Dataset [8], consisting of 3.5 million games collected from players of assorted skill, and the Kingbase Chess Dataset [9], consisting of 2.8 million games collected from the top 5000 players in the world.

Both of these datasets are comprised of text encodings of Chess known as Portable Game Notation (PGN). Represented in this way, the moves of a Chess game are encoded as a list of simple strings specifying which piece was moved where. Importantly, the game state is implicitly encoded in this language, but there is no explicit representation of the board. In order to determine what move comes next, the model must learn to implicitly construct the board state from these transitions, without any encoded prior knowledge. PGN also provides native support for commentary within brackets `{}`. An example of PGN with commentary is depicted below:

```
"1.  e4 e5 2.  Nf3 Nc6 3.  Bb5 a6 {This opening is called the Ruy Lopez.}"
```

To preprocess these datasets, we first removed extraneous information in the data, such as move numbers, player names, date, game location and ranking. Next, we limited game length to 256 total moves (128 per player), in order to constrain our model block size effectively. We found that 98.9% of the games fit this criteria.

3.2 Commentary Dataset

While commentary is supported in PGN, neither the Kingbase nor Kaggle datasets include game commentary. Given that no ready-to-use datasets exist, we adapted a custom-built web scraper built by Jhatami et al. [5] to obtain our own Chess commentary dataset. Ultimately, we scraped 11,700 commentated games from `gameknot.com`. The final dataset consists of 298,000 individual commented PGN strings, in the same format as described above.

Commentary data was preprocessed similarly to the original Chess data. We ensured that all PGN strings were able to be properly encoded into the block size limit, while removing extremely short and nonsensical commentary strings. We used the `langdetect` Python package to remove non-English commentary, and removed PGN strings that do not start from the beginning of the game. We found that around 86% of the commentated games remained after this preprocessing.

3.3 Supplementary English Dataset

In order to teach the model the nuances of the English language - including spelling and grammar - we selected a dataset of blog posts, collated by Schler et al. [10]. We found that the tone and writing style of the blog posts was in many respects similar to the commentary data we scraped. We preprocessed this dataset to extract raw paragraphs from the XML encodings, and once again used `langdetect` to select only entries that were in English. The final processed dataset consisted of 427,987 lines of English text.

4 Method

Given that our approach hinges on casting Chess as a sequential language problem, using deep sequence models makes a great degree of sense. The demonstrated efficacy of transformer models in the context of language, particularly given the natural interpretation of their attention mechanism on the progression of a Chess game, makes them a clear choice for us.

4.1 Baseline

For our baseline performance, we consider (and report, in our Experiments section) the results of Noever et al. [4] and ChePT-1 [7].

4.2 Transformer Models

Transformer models were first introduced in 2017 by Vaswani et al. in their paper 'Attention Is All You Need' [11]. This architecture improves traditional recurrent and convolutional neural networks architectures for sequence-to-sequence modelling by replacing these mechanisms entirely with attention (specifically multi-head attention). This has the advantage, among others, of making training highly parallelisable (alleviating the traditional major shortcoming associated with recurrent models) and therefore enabling the efficient training across huge corpora.

Multi-headed attention builds on traditional attention, which takes as input a set of queries and keys (of dimension d_k) and values of (dimension d_v) stacked into the matrices Q, K and V respectively, and computes as per equation 1. Multi-headed attention extends this idea by allowing a further dimension of learning to be applied to this mechanism, by learning `num_heads` different projections to be applied to the same input matrices. Mathematically, it computes using equations 2 and 3 below.

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

$$MultiHeadAttention(Q, K, V) = [head_1, ..., head_{num\_heads}]W^O \tag{2}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{3}$$

where the scaling factor of $\frac{1}{d_k}$ is applied to ensure that the softmax gradients do not become extremely small, and the matrices W are appropriately shaped learned parameter matrices.

The transformer model then uses this multi-headed attention to allow every position in the decoder to individually attend to all positions in the input sequence. Moreover every layer in the encoder and decoder respectively use this multi-headed attention mechanism to attend to all positions in their respective previous layers. In the context of Chess, this allows many different points in the encoding and decoding sequence of the model to 'keep track' of the previous Chess moves, giving transformers a massive advantage over recurrent models in their ability to learn to reconstruct the board space, and to perform inference while being highly sensitive to small changes in the input sequence.

4.2.1 Architectures

While transformer models are significantly more efficient to train than many other comparable sequence models, they are still incredibly computationally intensive. The final model that we present in this paper has been trained for around 144 hours on an NVIDIA 2080Ti graphics card. As such, we had to make architectural decisions relatively early in the development process, as changing the model later on would be extremely expensive given our limited resources. Transformer model architectures can vary

along a number of dimensions; specifically, the embedding size, the hidden size, the block size (maximum input sequence length), the number of layers, the vocabulary size, and the number of attention heads.

We performed early architectural experiments on a Chess-only task to reason about what potential model architectures would be appropriate (using the pretraining objective detailed in the 'Chess Pretraining' objective section). We analysed the following architectures (note that we had already settled on a block size of 256, as discussed in our data section):

Model Name	Total Parameter Count	Layers	Num Heads	Embed Size	Block Size
Baseline	11,509,760	8	16	256	256
Baseline-Large	60,821,504	16	32	512	256
Many-Heads	18,548,800	12	64	256	256
Noever et al. [4]	774,000,000	12	12	1032	1024
ChePT-1 Architecture [7]	9,600,000	12	16	256	512
ChePT-2 Architecture	14,668,800	12	32	256	256

Table 1: Considered model architectures

Given that we knew we were going to be asking far more of the model than just the 'Chess Pretraining' objective, we were looking for a model that clearly did not struggle with this task, but also a model that was not too complex so that we could feasibly train it on our limited resources. We found that the baseline architecture struggled to gain traction on the Chess Pretraining task, generally because it seemed to 'forget' input moves quite quickly - reacting only to more recent moves. As such, we increased the number of layers and number of heads, and eventually found what seemed to be a 'sweet spot' for both training tractability and task performance with the ChePT-2 Architecture as indicated.

4.2.2 Embeddings

Literature generally suggests that subword embedding provides optimal performance with transformer models. However, in addition to the English language, we also needed to encode every possible Chess mode (on the order of 10^4). The number of parameters in a transformer grows rapidly with the vocabulary size, and as such we were heavily resource constrained. We therefore settled on word-level embeddings for Chess moves (e.g. the PGN move 'e4' or 'O-O-O' is a single embedded word), and character level embeddings for English (reasoning that the set of Chess-related words would be sufficiently smaller than the set of all English words, to justify this decision). This yielded a vocabulary of 10011 total embeddings.

4.3 Training Objectives

4.3.1 Chess Pretraining

To imbue the model with an understanding of the rules of Chess, we used our larger and lower-skilled Kaggle dataset. We employed a next-word prediction pretraining objective, where the model learns to take preceding PGN, and predict the next move. By the nature of transformer models, we can parallelise this process across all moves in a game. We pad the embedded input vector to be equal to block size. We ran two epochs of pretraining with this objective (until performance converged), for a total of 7 million training iterations.

4.3.2 Chess Finetuning

Simply understanding the rules of Chess is insufficient, we also need to imbue the model with an understanding of strategy - particularly in the later-game, where skill gaps become more overt. As such, we applied the same pretraining objective, but this time using the higher-skilled Kingbase dataset. As such, we have a form of transfer-learning take place. The Kaggle dataset teaches the model the rules, and the model can then use Kingbase to focus on strategy. Moreover, we broke training examples down into early, mid and late-game, and focused training on the latter two, since early-game performance after pretraining was already quite good. Using this approached, we hoped to give the model more exposure to the comparatively data-sparse regime of late-game chess moves. We ran eight epochs of this finetuning, for a total of 22.4 million training iterations.

4.3.3 English Pretraining

Due to the relatively small size of the sample commentary dataset that we were able to scrape, it was clear that the model would struggle to learn both effective commentary, and acceptable spelling and sentence structure. Indeed, as documented in Mellsop, Swingle et al. [7], without any language support, this was certainly the case. As such, we decided to perform an English pretraining step as a form of 'transfer learning'. We used next-character prediction for block-size length sequences from the Blog Dataset (Supplementary English Dataset). We prefaced English sentences with a special masking character, so that the model would learn to predict English if and only if it was presented with this specific masking character. So as to mitigate the 'unlearning' of Chess-related knowledge, we interspersed this English pretraining randomly with Chess Finetuning examples. As such, the model had to continue to retain it's same Chess-based knowledge, while also predicting valid English sentences when required.

4.3.4 Commentary Finetuning

Once the model roughly understood both Chess gameplay and English, we needed to link these tasks together. To do so, we used the same next-character prediction objective, but integrated both PGN and English commentary. We allowed the first section of the input vector to be an encoded PGN as before. We then appended the masking character to force the model into 'English prediction' mode. We then appended the English commentary, followed by a final masking character and then padding until the vector length is equal to block size.

4.4 Training Hyperparameters

Due to VRAM constraints, we used a minibatch size of 10. We used a learning rate of $1e3$ for pretraining steps, and $1e4$ for finetuning steps. Once again, due to the computational complexity of training large transformer models, we were resource-constrained in exploring this hyperparameter space further, and as such relied on figures as presented in literature [11].

5 Experiments

For experiments and evaluation, we had our model (at various points in the training process), and the original ChePT-1 model each play 2000 newly-generated games against Stockfish 13 [12]. We include key statistics from those games alongside results presented by Noever et al. Crucially, these games are distinct from those seen by the model in the training dataset.

5.1 Game Statistics and Lengths

When ChePT-2 generates a move, we use the `python-chess` API to check that the move is legal given the current board state. If the move is not legal, we allow ChePT-2 to re-generate moves up to 5 times before deeming that it is unable to generate a legal move, and resign. As such, the model may attempt to make illegal moves that do not force resignation. Thus, the illegal move percentage captures the percentage of the time that ChePT-2 generates an illegal move, but does not necessarily result in a loss. Furthermore, we define the length of a game as the number of total moves made by the bot (known as half-moves). As such, the 'total number' of moves in a game is twice this number.

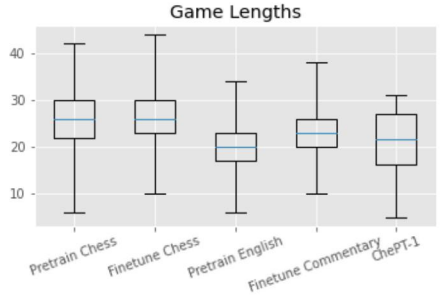

Figure 1: ChePT-2 Game Length Statistics

Model/Training Step	Num Wins	Num Losses	Num Draws	Illegal Move %
ChePT-2 Pretrain Chess	0	1916	84	2.6%
ChePT-2 Finetune Chess	0	**1897**	**103**	**2.1%**
ChePT-2 Pretrain English	0	1993	7	6.3%
ChePT-2 Finetune Commentary†	0	1931	69	3.1%
Noever et al. [4]	-	-	-	10%
ChePT-1 [7]	0	1967	33	3.9%

Table 2: Game outcomes (Model vs. Stockfish 13) †Final ChePT-2 Model

5.2 Normalised Centipawn Loss

While the above metrics provide a measure of aggregate model performance, we are also interested in analysing how the model's performance tracks through various game stages. As such, we use the Centipawn [13] move analysis metric as computed by Stockfish 13. We are particularly interested in how our model's performance compares to the 'ideal' move, which we assume to be that chosen by Stockfish 13 [12]. Across 2000 games, we plot the difference between the Centipawn score of the ideal move and the move that our bot actually played. The magnitude of this difference shows the extent to which Stockfish 13 outperformed our model.

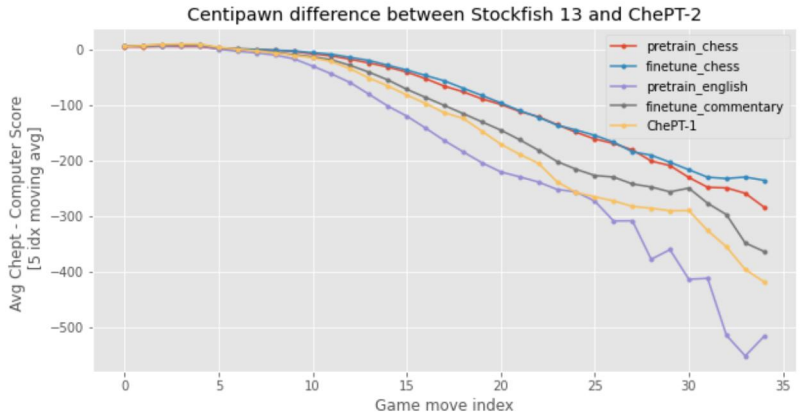

Figure 2: ChePT-2 Centipawn Score - Stockfish-13 Centipawn Score Across Move Indices (mean across 2000 games)

5.3 Commentary Analysis

Analysis of commentary proved difficult to automate. We considered using BLEU scores, similiar to those presented by Jhamtani et al. [5] and Zang et al. [6]. However, as discussed by in literature, the appropriateness of a BLEU score (a score originally designed for translation tasks) to the relatively unconstrained commentary space is limited [7] [14].

As such, we turned to other analysis techniques. Namely, we first found that (using the `pyspellchecker` Python package), that in over 500 fully commentated games between Stockfish 13 and ChePT-2, the generated commentary consisted of 96.9% valid English words. This represents a significant improvement over the findings of ChePT-1 [7], where spelling and grammar was a significant shortcoming.

Presented in Figures 3, 4, 5 and 6 are an assortment of Chess board configurations (based on proceeding PGN generated by the model playing against a human, not shown for the sake of brevity) and commentary predictions.

Figure 3: Predicted Commentary: the sicilian defense.

Figure 4: Predicted Commentary: a slight attack and trying to get him to b4!

Figure 5: Predicted Commentary: letting the bishop out we get pressure in the table.

Figure 6: Predicted Commentary: develops and undermines white knight.

5.4 Discussion

From our analysis metrics, it is clear that ChePT-2 yields significantly better Chess performance than both Noever et al. [4] and ChePT-1 [7]. With a significantly lower illegal move percentage, longer average games, and a higher draw rate against Stockfish 13, this is a sound conclusion. We can credit this advancement in part to a more sophisticated model architecture than ChePT-1 (although Noever et al. use a full GPT-2 architecture, which is far larger than ChePT-2). Moreover, ChePT-2's use of move-level embeddings reduces the number of preceding items that the attention mechanism needs to attend to, likely contributing to this improvement.

Additionally, as per Figure 2, we see that ChePT-2 remains stronger for longer over the course of games relative to ChePT-1 (which already outperformed Noever et al. in late game stages [7]). However, the discrepancy between Stockfish 13 and ChePT-2 as measured by the Centipawn score increases over the game progression, indicating that ChePT-2 continues to struggle more in the late game. Intuitively, this makes sense. Late game situations will be found more sparsely in the dataset than early games–giving ChePT-2 less late game experience. While Figure 2 shows our finetuning approach (focused on mid-to-late game) helped mitigate this, we still find ourselves limited in this regard. We note that, probably due to this factor, and much like Noever et al. and ChePT-1, ChePT-2 is never able to win against Stockfish 13. This is unsurprising; Stockfish 13 is a state-of-the-art Chess engine that has been under development for decades, and has numerous advantages over our system (explicit rules lists, stateful board encodings, etc.). As such, we are extremely pleased that our final system was able to reach draw conditions against Stockfish 13 around 5% of the time.

Moreover, ChePT-2 yields a significant commentary improvement over ChePT-1. Spelling and grammar is strong, with solid sentence structure and very few spelling errors. Commentary is almost always punctuated appropriately, and relative to the findings by Mellsop, Swingle et al. [7] manual analysis indicates superior language ability, and a fundamentally better understanding of Chess. We find that the model is able to accurately commentate a number of the more common 'book openings' such as demonstrated in Figure 3 [15]. Additionally, it demonstrates foresight into how its moves will be responded to (Figures 4 and 6), and how its moves will enable it to engage subsequently (Figures 5 and 6). Based on these examples and others that we have observed, there is clear Chess insight demonstrated by the model's commentary.

One interesting factor to discuss is the reduction in Chess performance between the 'Finetune Chess' training step, and the final ChePT-2 model (post 'Finetune Commentary' step), as per Figure 2 and Table 2. This decrease in performance makes sense intuitively: we are asking the chess-specialized model to suddenly take on and perform an almost entirely different task. As such, the model's capacity for learning will have to be 'divided' between these two tasks. Even so, we note that the model's Chess knowledge persists for the most part. We would expect this discrepancy to decrease with a sufficiently large model.

6 Conclusion & Future Work

In conclusion, ChePT-2 succeeds at building significantly upon the results of ChePT-1 and Noever et al. Our increase in Chess performance over and above both models, in addition to our significantly improved commentary - particularly with respect to syntax and grammar - mean that ChePT-2 is a significant advancement.

Moreover, the success of this model raises fascinating questions about how it might be applied to other state-based problem spaces - an area that has traditionally been dominated by reinforcement learning. ChePT-2 might be analysed and built in the context of Chess, but if we take a step back, what we have really built here is a versatile framework for unsupervised learning and self-commentary in state space. With an encoding of state transitions and a corpus of commentated samples, this framework could excel in a more general application while providing a degree of insight into its decision-making.

To build on our current work, we would like to explore utilising different architectures. As described earlier in this paper, the computational complexity of models in this domain prevented us from running comparative explorations between architectures. Moreover, the fact that our final model's performance decreased slightly when commentary was introduced perhaps indicates that the architecture we selected did not quite have the learning capacity to become as effective as possible at both Chess and commentary.

Additionally, our current training strategy is innately limited by the human-generated Chess data that we have. We would like to explore adversarial training, whereby our model would play games against Stockfish, while we record the 'ideal' move at each point. We could then train the model on this generated data, potentially allowing the model to learn far beyond just the human datasets we have currently.

Finally, we would benefit from a significantly larger and better Chess commentary dataset. No ready-to-use datasets in this domain exist, and as discussed our dataset was obtained using a custom-built web crawler. A larger and better Chess commentary dataset would likely boost our commentary insight and performance significantly.

References

[1] IEEE. How IBM's Deep Blue Beat World Champion Chess Player Garry Kasparov. *https://spectrum.ieee.org/the-institute/ieee-history/how-ibms-deep-blue-beat-world-champion-chess-player-garry-kasparov*, 2021.

[2] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017.

[4] David Noever, Matt Ciolino, and Josh Kalin. The Chess Transformer: Mastering Play using Generative Language Models. *arXiv preprint arXiv:2008.04057*, 2020.

[5] Harsh Jhamtani, Varun Gangal, Eduard Hovy, Graham Neubig, and Taylor Berg-Kirkpatrick. Learning to generate move-by-move commentary for chess games from large-scale social forum data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1661–1671, 2018.

[6] Hongyu Zang, Zhiwei Yu, and Xiaojun Wan. Automated Chess Commentator Powered by Neural Chess Engine. *arXiv preprint arXiv:1909.10413*, 2019.

[7] Henry Mellsop, Colton Swingle, and Alex Langshur. ChePT - Applying Deep Neural Transformer Models to Chess Prediction and Self-Commentary. *CS224N Final Projects*, 2021.

[8] Milesh1. 3.5 Million Chess Games. *kaggle.com/milesh1/35-million-chess-games*, 2018.

[9] Collated by Kingbase. 2.8 Million Collated 2000+ MMR Chess Games. *www.kingbase-chess.net*, 2017.

[10] J Schler, S Argamon, and J Pennebaker. Effects of Age and Gender on Blogging. *Proceedings of the 2006 AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs*, 2006.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *31st Conference on Neural Information Processing Systems*, 2017.

[12] Stockfish. Stockfish 13. *stockfishchess.org/blog/2021/stockfish-13/*.

[13] WikiChess. Centipawn Loss. *chess.fandom.com/wiki/Centipawn*.

[14] Jekaterina Novikova and Verena Rieser. Why we need new evaluation metrics for NLG. *Proceedings of 2017 Conference on Empirical Methods in NLP*.

[15] Chess.com. Chess Book Openings. *www.chess.com/openings*.

As described, we would also like to make once again explicit the connection to Henry Mellsop, Colton Swingle, and Alex Langshur's CS224N project from Winter 2021. The inspiration was similar, and we of course applied learnings from this project and sought to significantly improve on our findings - which we did. However, except for the transformer model implementation code which was provided by CS224N, no code is shared between that project and this project. The approach, except for the model architecture itself, is completely different - from the embeddings, through to the training objectives and the evaluation metrics and pipelines.

7 Contributions

Harry built out the Chess and English data pipelines, and a lot of the training objective function logic. He built out evaluation functionality to play the bot against Stockfish and analyse the results.

Colton built out the web crawler to obtain and pre-process the commentary dataset. He also worked on the training objective logic.

Collin helped build the evaluation logic and plotting infrastructure through the notebooks. He managed a lot of the model training and helped develop the infrastructure around saving/loading checkpoints.

While the above is a broad-brush 'division' of work, we all helped each other out and collaborated significantly on all areas of the project. We all contributed equally to the various writeups throughout the project process.

8 Source Code

`https://github.com/HarryMellsop/chept-2`