Synthetic Languages for democratizing Language Model Research

Dhruv Dhamani DDHAMANI@CHARLOTTE.EDU
Narges Zare NZARE@CHARLOTTE.EDU
University of North Carolina at Charlotte, Charlotte, NC, USA

Abstract

Language models are crucial for natural language processing tasks, relying on predicting the next word from a given sequence of words. However, training these models to incorporate reasoning and logic is challenging, especially on consumer-grade GPUs, which have limited processing power. This report introduces a synthetic language inspired by the game of Blackjack to support language model research on more accessible hardware. We discuss the motivation for using synthetic languages, detail the development of our Blackjack-inspired synthetic language, and present results showing steady improvements in model accuracy and reductions in error as training advances. We also explore the impact of context window size on the model's performance across different game configurations. This approach aims to make language model research more feasible for a wider range of researchers by using a simpler and more manageable platform.

1. Introduction

Language modeling involves learning a "model" of any language and using this model to predict the next word in a sequence. However, to accurately predict the next word, the model needs to understand not only the language's syntax but also its semantics and logic. This can be particularly challenging for language models trained on consumer GPUs, as they have limited capacity to learn complex patterns and relationships. To address this challenge, we developed a synthetic language of configurable complexity inspired by the game of Blackjack. This synthetic language essentially represents a transcript of the game of Blackjack. The model can be asked to "play" by sharing a transcript of the transpired events, and using its next word prediction of hit or stand as an action.

2. Motivation and Background

Large Language Models (LLMs) are increasingly powerful but require significant computational resources in terms of processing speed and memory. This is primarily due to their complex design, which includes a vast number of parameters and the use of self-attention, a mechanism crucial for their functionality but computationally intensive.

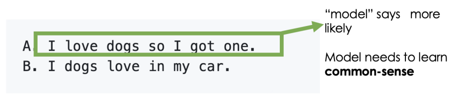
The high computational requirements pose a substantial barrier to entry for researching and developing new LLM architectures, particularly for those without access to extensive computing resources. One solution to this challenge is the use of synthetic languages, which are simpler and computationally cheaper to work with. These artificial languages are designed to mimic some of the core tasks of natural language processing but do not require the model to learn and store extensive real-world knowledge. For example, unlike natural languages, predicting the next word in a synthetic language does not require the model to remember factual.

Research, such as the study by Arora et al. (2023), shows that synthetic languages can effectively predict the performance of LLMs on more complex natural languages. In this particular study, tasks involving Associative Recall in synthetic languages were found to predict a significant portion of the performance differences observed between models using self-attention and those without it when trained on a large dataset known as The Pile (Gao et al. (2020)). This insight supports further development in LLM architectures and indicates that simpler, synthetic tasks can be valuable tools in advancing language model research. Moreover, the "Synthetics for Language Modeling" initiative from Hazy Research (2024) provides additional examples of how synthetic languages are being employed to foster advancements in this field. This approach not only makes LLM research more accessible but also speeds up the process of understanding and improving these complex systems.

2.1. Language Modeling

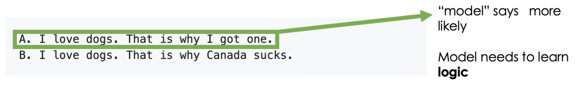
Effective language models require more than just an understanding of syntax and structure; they must also grasp common-sense knowledge and logic to generate meaningful sentences. Previous research, exemplified by Allen-Zhu and Li (2024)'s exploration of knowledge capacity scaling laws, emphasizes the necessity for models to efficiently learn both facts and logic. Additionally, studies on associative recall underscore the importance of contextual comprehension and the ability to recall associated tokens (Research (2023)).

For instance, consider figure 1, where the model must recognize common-sense knowledge: understanding that individuals fond of dogs are likely to adopt one as a pet. Similarly, in figure 2, the model's prediction of sentence 'A' as logical reflects its understanding of the connection between loving dogs and adopting one, contrasting with the illogical statement in sentence 'B'. These examples highlight that a language model's coherence relies not only on its grasp of language structure but also on its integration of common-sense knowledge and logical reasoning. Furthermore, the depth of knowledge required directly correlates with the number of parameters a model needs (Allen-Zhu and Li (2024)), where a language model can learn approximately 2 bits of knowledge per parameter.



```
A  I love dogs so I got one.
B. I dogs love in my car.
```

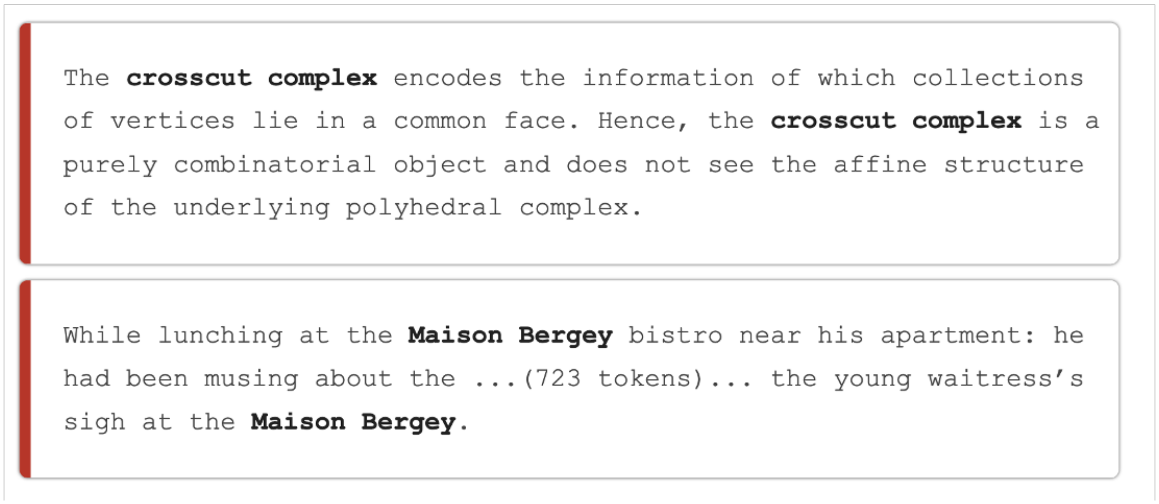
Figure 1: Common-sense knowledge in language modeling.



```
A. I love dogs. That is why I got one.
B. I love dogs. That is why Canada sucks.
```

Figure 2: Logical connections in language modeling.

Associative recall is a crucial aspect of language modeling that requires different skillsets to learn. It refers to a language model's ability to recall associated tokens from the context or prior history. Figure 3 demonstrates how measuring a model's performance in predicting repeated bigrams, such as "crosscut complex" and "Maison Bergey," through "Associative Recall (AR) Hits" can help assess its effectiveness in processing these repeated phrases (Research (2023)). This approach uses perplexity on AR Hits as a proxy to evaluate the model's handling of context and memory, which is essential for achieving human-like language understanding.



The **crosscut complex** encodes the information of which collections of vertices lie in a common face. Hence, the **crosscut complex** is a purely combinatorial object and does not see the affine structure of the underlying polyhedral complex.

While lunching at the **Maison Bergey** bistro near his apartment: he had been musing about the ...(723 tokens)... the young waitress's sigh at the **Maison Bergey**.

Figure 3: Evaluating associative recall using perplexity on AR Hits. It's taken from Research (2023).

Figure 4 illustrates the difference between Associative Recall and Multi-query Associative Recall (MQAR) tasks. In the Associative Recall task, the model is given a key (A) and is expected to output the corresponding value (3). In contrast, the MQAR task involves providing the model with multiple keys (A, C) and expecting it to output their corresponding values (3, 8) (Research (2023)). MQAR better captures the quality gaps between synthetic and real-world data compared to prior AR formulations, as language modeling often requires performing multiple recalls in a single forward pass, at varying positions, with tokens from a large vocabulary.

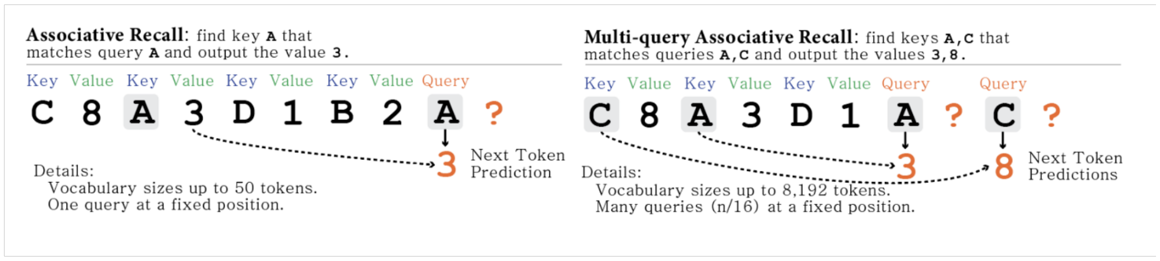

Associative Recall: find key A that matches query A and output the value 3.
Key Value Key Value Key Value Key Value Query
C 8 A 3 D 1 B 2 A ?
3
Next Token Prediction
Details:
Vocabulary sizes up to 50 tokens.
One query at a fixed position.
Multi-query Associative Recall: find keys A,C that matches queries A,C and output the values 3,8.
Key Value Key Value Key Value Query Query
C 8 A 3 D 1 A ? C ?
3
8
Next Token Predictions
Details:
Vocabulary sizes up to 8,192 tokens.
Many queries (n/16) at a fixed position.

Figure 4: Comparison of Associative Recall and Multi-query Associative Recall tasks. It's taken from Research (2023).

Figure 5 shows the performance of different language models (Attention, BaseConv, Based, and Mamba) on the MQAR task across various sequence lengths (64, 128, 256, and 512) (Research (2023)). The accuracy of the models is plotted against the model dimension. The results indicate that the Attention model outperforms the other models, particularly at longer sequence lengths, demonstrating its superiority in handling context and memory. These findings suggest that a synthetic language can be created to require the model to learn how to perform associative recall (Arora et al. (2023)).

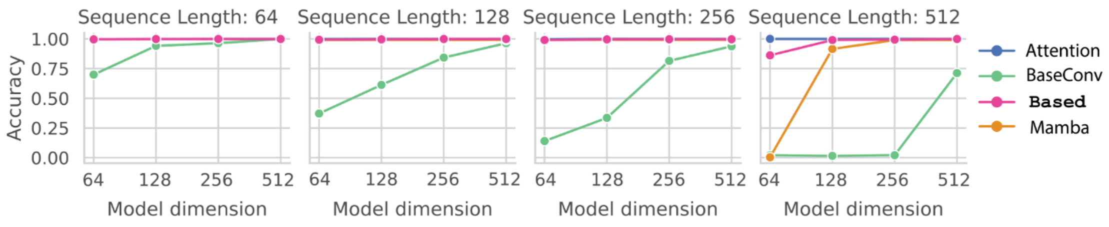
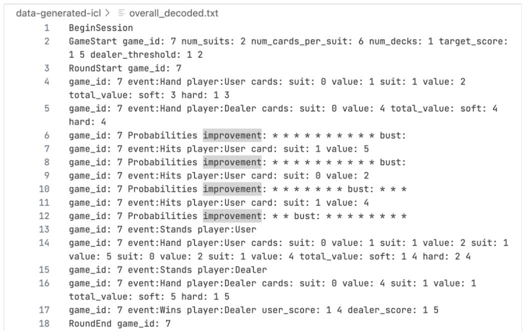
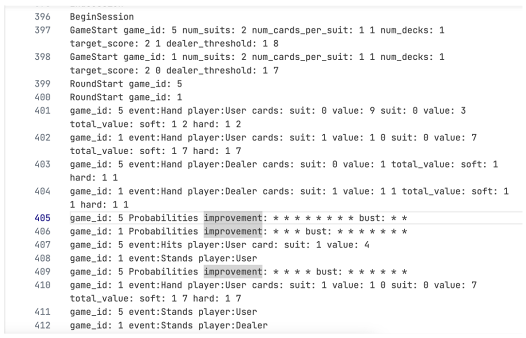


Figure 5: Performance of language models on the MQAR task across different sequence lengths. It's taken from Research (2023).



```
BeginSession
GameStart game_id: 7 num_suits: 2 num_cards_per_suit: 6 num_decks: 1 target_score: 1 5 dealer_threshold: 1 2
RoundStart game_id: 7
game_id: 7 event:Hand player:User cards: suit: 0 value: 1 suit: 1 value: 2 total_value: soft: 3 hard: 1 3
game_id: 7 event:Hand player:Dealer cards: suit: 0 value: 4 total_value: soft: 4 hard: 4
game_id: 7 Probabilities improvement: * * * * * * * * * * bust:
game_id: 7 event:Hits player:User card: suit: 1 value: 5
game_id: 7 Probabilities improvement: * * * * * * * * * * bust:
game_id: 7 event:Hits player:User card: suit: 0 value: 2
game_id: 7 Probabilities improvement: * * * * * * * bust: * * *
game_id: 7 event:Hits player:User card: suit: 1 value: 4
game_id: 7 Probabilities improvement: * * bust: * * * * * * * *
game_id: 7 event:Stands player:User
game_id: 7 event:Hand player:User cards: suit: 0 value: 1 suit: 1 value: 2 suit: 1 value: 5 suit: 0 value: 2 suit: 1 value: 4 total_value: soft: 1 4 hard: 2 4
game_id: 7 event:Stands player:Dealer
game_id: 7 event:Hand player:Dealer cards: suit: 0 value: 4 suit: 1 value: 1 total_value: soft: 5 hard: 1 5
game_id: 7 event:Wins player:Dealer user_score: 1 4 dealer_score: 1 5
RoundEnd game_id: 7
```



```
BeginSession
GameStart game_id: 5 num_suits: 2 num_cards_per_suit: 1 1 num_decks: 1 target_score: 2 1 dealer_threshold: 1 8
GameStart game_id: 1 num_suits: 2 num_cards_per_suit: 1 1 num_decks: 1 target_score: 2 0 dealer_threshold: 1 7
RoundStart game_id: 5
RoundStart game_id: 1
game_id: 5 event:Hand player:User cards: suit: 0 value: 9 suit: 0 value: 3 total_value: soft: 1 2 hard: 1 2
game_id: 1 event:Hand player:User cards: suit: 1 value: 1 0 suit: 0 value: 7 total_value: soft: 1 7 hard: 1 7
game_id: 5 event:Hand player:Dealer cards: suit: 0 value: 1 total_value: soft: 1 hard: 1 1
game_id: 1 event:Hand player:Dealer cards: suit: 1 value: 1 1 total_value: soft: 1 1 hard: 1 1
game_id: 5 Probabilities improvement: * * * * * * * * bust: * *
game_id: 1 Probabilities improvement: * * * bust: * * * * * * *
game_id: 5 event:Hits player:User card: suit: 1 value: 4
game_id: 1 event:Stands player:User
game_id: 5 Probabilities improvement: * * * * bust: * * * * * *
game_id: 1 event:Hand player:User cards: suit: 1 value: 1 0 suit: 0 value: 7 total_value: soft: 1 7 hard: 1 7
game_id: 5 event:Stands player:User
game_id: 1 event:Stands player:Dealer
```

Figure 6: Example game session showing a single Blackjack game with 2 suits and 6 cards per suit, using 1 deck.

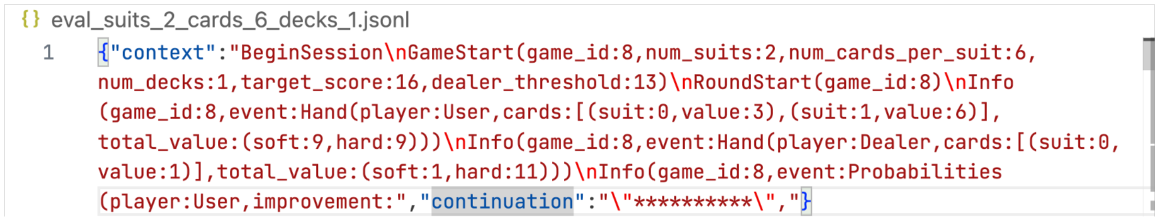
Figure 7: Configuration simulating two concurrent Blackjack games using 2 suits with 1 card per suit and 1 deck each.

2.2. Synthetic Language Inspired by Blackjack

Our initial idea was that Multi-Query Key-Recall tasks are too easy, necessitating a more challenging synthetic task to adequately evaluate natural language generation (NLG) models' capabilities. Simply demonstrating the ability to recall based on a single key is insufficient; models must learn to infer from multiple contextual "keys" and perform logical reasoning. To address this, we proposed a synthetic language inspired by the game of Blackjack that challenges language models to perform associative recall and logical reasoning.

The idea for configuring complexity involved varying parameters such as the number of suits, cards per deck, and decks used, enabling the simulation of multiple simultaneous games with diverse contexts. This configurability allows for the creation of tasks with varying difficulty levels, catering to different research requirements and model capabilities. Additionally, we aimed for straightforward model evaluation using this synthetic language, where the model is provided with the game context up to a specific point marked by "improvement:", and then asked to predict probabilities based on the given context, generating stars for improvement with 10 being the maximum probability. Figures 6 and 7 illustrate two different configurations of our model.

This approach assesses the model's ability to infer from various contextual cues and perform logical reasoning. Furthermore, to ensure accessibility, we designed a model architecture suitable for training on consumer-grade hardware. Specifically, a 15-decoder layer transformer model with two self-attention heads, a 4096 context length, and a batch size of 8 can be trained on a consumer GPU with 22GB VRAM. With approximately 27 million parameters, this model is feasible without access to high-end computational resources.



```
{} eval_suits_2_cards_6_decks_1.jsonl
{"context":"BeginSession\nGameStart(game_id:8,num_suits:2,num_cards_per_suit:6,
num_decks:1,target_score:16,dealer_threshold:13)\nRoundStart(game_id:8)\nInfo
(game_id:8,event:Hand(player:User,cards:[(suit:0,value:3),(suit:1,value:6)],
total_value:(soft:9,hard:9)))\nInfo(game_id:8,event:Hand(player:Dealer,cards:[(suit:0,
value:1)],total_value:(soft:1,hard:11)))\nInfo(game_id:8,event:Probabilities
(player:User,improvement:","continuation":"\"**********\","}
```

Figure 8: Example game context provided as input to the language model, containing the game ID, number of suits and cards, target score, dealer threshold, initial hands dealt to the player and dealer, and total hand values. The model's continuation predicting the probability of improvement (represented by "**********") is shown, which was later grouped with other continuations by predicted next move for analysis and improvement of probability predictions across different game trajectories.

3. Methodology

To evaluate our proposed synthetic language inspired by the game of Blackjack, we followed a structured methodology:

Dataset Generation: We generated 6000 game sessions of a synthetic Blackjack-inspired language for the training dataset and an additional 600 sessions with the same configurations for the evaluation dataset. The player actions were decided using card-counting strategies. The configurations varied from 2 suits with 6 cards per suit to 4 suits with 15 cards per suit, using 1 deck. Additionally, we expanded the range of game configurations in our new experiments, varying the number of suits (2, 3, 4), cards per suit (6 to 15), and context window sizes (2048, 4096, 8192) to investigate their impact on model performance.

The average game session length was around 2500 tokens, with the longest reaching 7200 tokens.

Model Architecture and Thesis: Our thesis was that understanding the entire game context thus far is necessary to accurately count cards and predict probabilities, making this a good synthetic test for long-context language modeling. To accommodate this, we trained transformer language models with varying context window sizes (2048, 4096, 8192) to examine the effect of context length on model performance.

Evaluation Methodology: For evaluation, we computed the perplexity on the eval set and extracted all instances where the model predicted probabilities, grouping them by game configuration into a JSONL dataset. We then computed the accuracy of the model in predicting the correct number of stars (representing the probability of improvement) for each configuration. The model was evaluated by providing example contexts containing the game state up to a point, followed by the model's continuation predicting the probability of improvement. These continuations were grouped by predicted next moves, with the goal of analyzing and improving the model's probability predictions for different possible game trajectories and contexts. Additionally, we reported accuracy and the mean star error, calculated as the mean absolute difference between the predicted and target improvement in stars. In our new experiments, we introduced additional visualizations, such as line charts comparing mean error across checkpoints for different context window sizes (Image 2) and a heatmap displaying mean error for the checkpoint at 6000 batches across various context windows and deck sizes (Image 1).

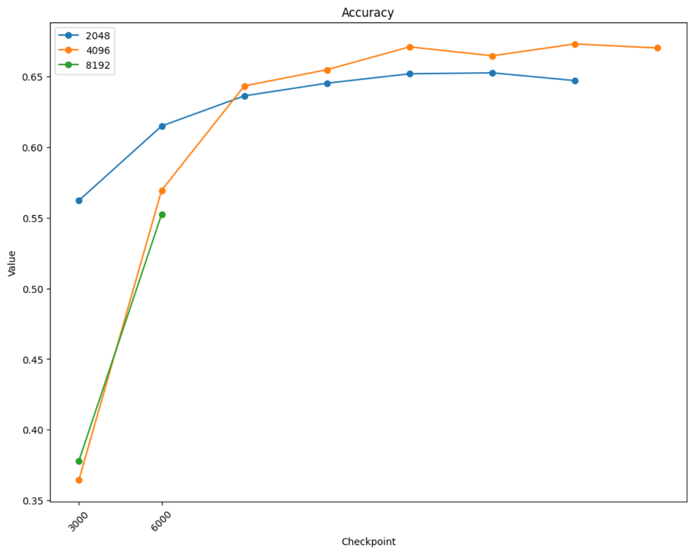


Figure 9: **Accuracy Comparison for Different Checkpoints** - Line chart showing the accuracy of the language model in predicting probabilities of improvement on the synthetic Blackjack task, evaluated at checkpoints from 3000 to 24000 batches of training data across different context window sizes (2048, 4096, 8192). The model's accuracy generally improves with more training. *Not all checkpoints finished training in time, only available checkpoints were evaluated.*

4. Results

The performance of our transformer language model on the synthetic Blackjack task was evaluated at several checkpoints during training, ranging from 3,000 batches to 24,000 batches. The model's objective was to continue Blackjack game transcripts by accurately predicting the probabilities of improving the player's hand given the current game state.

As illustrated by Figure 9, the model's average accuracy steadily improved as it was trained on more data. This lends credibility to our synthetic language as a valid task.

We also compute "mean star error", which is the absolute difference in the number of "stars" predicted by the model vs the true value. The mean error for each model and checkpoint can be found in Figure 10.

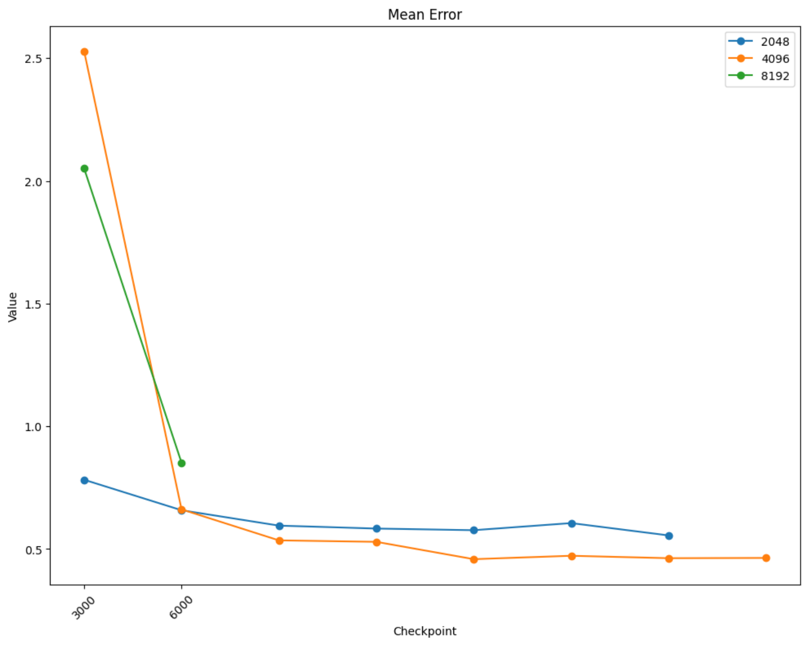


Figure 10: **Mean Error Comparison for different context lengths** - depicting the mean error between the model's predicted score improvement probabilities and target values, plotted against number of gradient update steps on the x-axis.

To further investigate the impact of context length and game configuration on the model's performance, we generated a heat map (Figure 11) comparing the mean error for various context lengths and deck sizes at a specific checkpoint (6000 gradient update steps). This heat map provides valuable insights into the model's behavior across different scenarios.

The heat map reveals that the model's performance varies depending on the context length and the size of the deck. Generally, the model achieves lower mean errors with larger context lengths, suggesting that having access to more contextual information improves the model's ability to make accurate predictions. This highlights the validity of our synthetic Blackjack language as a long-range generative model benchmark.

Moreover, the heat map shows that the model's performance is influenced by the size of the deck, which represents the complexity of the game configuration. The model tends to have higher mean errors for larger deck sizes, indicating that more complex game configurations pose a greater challenge for the model. This finding emphasizes the need to evaluate language models on a range of task difficulties to assess their robustness and generalization capabilities.

The insights gained from the heat map analysis contribute to our understanding of the factors that affect the model's performance on our synthetic Blackjack task. By considering the interplay between context length and game complexity, we can make informed decisions when designing and optimizing language models for similar tasks. Additionally, the heat map serves as a valuable tool for identifying areas where the model struggles and can guide future research efforts in improving the model's performance across diverse scenarios.

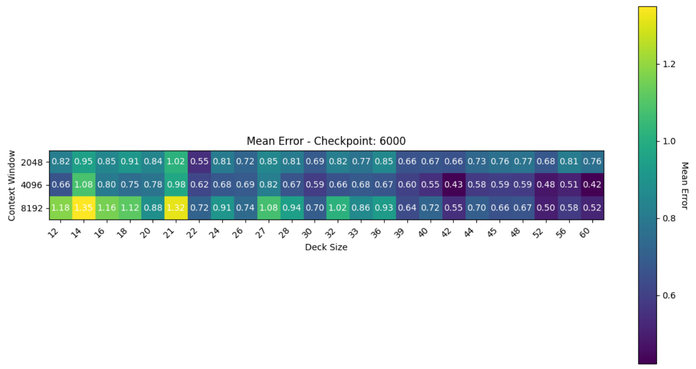


Figure 11: **Mean Error Heat Map** - comparing the mean error for various context lengths, for various game configurations (depicted by size of the deck) for a model checkpoint after 6000 gradient update steps. The heat map reveals the impact of context length and game complexity on the model's performance, with larger context lengths generally leading to lower mean errors, and larger deck sizes posing a greater challenge for the model. *Comparision isn't very useful at only 6000 steps, but shows the significance of future work.*

5. Summary

In this project, we developed a synthetic language data generator based on the game of Blackjack to enable language model research on consumer-grade hardware. Recognizing the limitations of Multi-Query Key-Recall tasks for evaluating natural language generation capabilities, we proposed a more challenging synthetic task that requires logical reasoning and inference from multiple contextual cues.

Our approach involved generating Blackjack game transcripts with configurable complexity by varying parameters such as the number of suits, cards per deck, and decks used. This allowed us to simulate diverse game contexts across varying difficulty levels, as outlined in our project proposal. To facilitate this research, we created a data generator and made it publicly available on GitHub, along with the generated training and evaluation datasets.

We trained transformer language models with different context window sizes (2048, 4096, 8192) on the generated Blackjack data. The heat map analysis provided valuable insights into the factors affecting the models' performance, particularly the interplay between context length and game complexity. Larger context lengths generally led to lower mean errors, highlighting the validity of our synthetic Blackjack language as a long-range generative model benchmark. Additionally, the heat map revealed that larger deck sizes, representing more complex game configurations, posed a greater challenge for the models.

Our work demonstrates the potential of using synthetic languages, such as the Blackjack-inspired language, to enable language model research on consumer-grade hardware. By providing a challenging task that requires logical reasoning and inference from multiple contextual cues, our approach offers a valuable platform for advancing natural language generation capabilities and democratizing access to language model research.

1. Comparing the performance of scaled-down language models on this task with the performance of scaled-up language models reported in the literature to assess the predictive power of our synthetic language benchmark.

2. Investigating how varying the parameters of the language models affects their performance on this task, providing insights into model architecture design and optimization.

3. Exploring alternative model architectures, such as attention-free models or novel architectures (e.g., BitNet), to evaluate their effectiveness on the synthetic Blackjack task and compare their performance with self-attention-based models.

References

Z. Allen-Zhu and Y. Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv*, Apr 2024. doi: 10.48550/arXiv.2404.05405.

S. Arora et al. Zoology: Measuring and improving recall in efficient language models. 2023.

L. Gao et al. The pile: An 800gb dataset of diverse text for language modeling. 2020.

Hazy Research. AISys Building Blocks. https://github.com/HazyResearch/aisys-building-blocks, 2024. [Online; accessed 5-May-2024].

S. Ma et al. The era of 1-bit llms: All large language models are in 1.58 bits. *Journal Name Here or Leave Blank*, 2024.

Hazy Research. Zoology1 analysis, December 2023. Available online: https://hazyresearch.stanford.edu/blog/2023-12-11-zoology1-analysis [Accessed May 5, 2024].

H. Wang et al. Bitnet: Scaling 1-bit transformers for large language models. *Journal Name Here or Leave Blank*, 2023.

Appendix A. Synthetic Blackjack Data Generator

A.1. Generic Blackjack Variant

To create a synthetic language dataset suitable for training and evaluating language models, we developed a generic variant of Blackjack designed to ensure the player must make at least one hit or stand decision on average per game. This is achieved by dynamically calculating the target score and dealer threshold based on the game configuration parameters, such as the number of suits, cards per suit, and decks used.

The target score and dealer threshold are computed using the following algorithm:

Algorithm 1: Calculate Thresholds

```
Data: num_suits, num_cards_per_suit
Result: target_score, dealer_threshold
max_card_value = min(num_cards_per_suit, 10);
num_face_cards = num_suits * max(0, num_cards_per_suit - 10);
total_value = sum(range(2, max_card_value+1)) * num_suits + num_face_cards *
 10 + num_suits * 11;
avg_card_value = total_value / (num_suits * num_cards_per_suit);
target_score = round(3.0 * avg_card_value);
dealer_threshold = round(2.5 * avg_card_value);
return target_score, dealer_threshold;
```

In Blackjack, both the player and the dealer get assigned two cards at the beginning of each round. By setting the target score to three times the average card value, we ensure the player needs to make at least one useful hit/stand decision when playing the game.

The target score and dealer threshold are randomly varied by +1, +0, -1. This ensures the model cannot simply "memorize" these values in it's, and that it needs to learn to "attend" to them from context.

A.2. Probability Calculation

During each round, the player's decision to hit or stand is based on the probabilities of improving their hand without exceeding the target score (bust). These probabilities are computed using the following algorithm:

The algorithm iterates through all possible card values, calculating the probability of drawing each card based on the remaining cards in the deck. For each potential card, it simulates adding it to the player's hand and evaluates the resulting hand value against the target score. If the new hand value exceeds the target score, it contributes to the bust probability; otherwise, it adds to the improvement probability. The player then makes a decision based on whether the improvement probability is higher than the bust probability.

No. of Suits	No. of Cards per Suit	Target Score	Dealer Threshold
2	6	16	13
2	7	16	14
2	8	17	14
2	9	18	15
2	10	20	16
2	11	20	17
2	12	21	18
2	13	22	18
2	14	23	19
2	15	23	19
3	6	16	13
3	7	16	14
3	8	17	14
3	9	18	15
3	10	20	16
3	11	20	17
3	12	21	18
3	13	22	18
3	14	23	19
3	15	23	19
4	6	16	13
4	7	16	14
4	8	17	14
4	9	18	15
4	10	20	16
4	11	20	17
4	12	21	18
4	13	22	18
4	14	23	19
4	15	23	19

Table 1: Target scores and dealer thresholds for different game configurations in the synthetic Blackjack variant.

A.3. Data Generation and Tokenization

The data generator produces a specified number of game sessions for each valid configuration of suits, cards per suit, and number of decks. These sessions are recorded in a text format, with each game event represented by a specific syntax.

The generated data is then tokenized using a custom tokenizer implemented in Rust, leveraging the `tokenizers` library. The tokenizer performs the following steps:

Algorithm 2: Calculate Probabilities

```
Data: player_hand, deck, target_score
Result: bust_prob, improvement_prob
remaining_cards = sum(deck.card_counter.counts);
bust_prob = 0;
improvement_prob = 0;
for card_value in range(1, deck.num_cards_per_suit+1) do
    prob = deck.card_counter.get_count(card_value) / remaining_cards;
    new_player_hand = player_hand + [Card{suit: 0, value: card_value}];
    hand_value = calculate_hand_value(new_player_hand);
    if hand_value > target_score then
        bust_prob += prob;
    else
        improvement_prob += prob;
    end
end
return bust_prob, improvement_prob;
```

1. **Trainer Configuration**: The `WordLevelTrainerBuilder` is used to configure the tokenizer trainer. It sets the vocabulary size to 9999, specifies a special `[PAD]` token, and sets the minimum frequency threshold to 0.

2. **Tokenizer Builder**: The `TokenizerBuilder` is used to construct the tokenizer. It uses the `WordLevel` model with an `[UNK]` token for unknown words.

3. **Normalization**: The tokenizer applies a sequence of normalizers to preprocess the text. This includes stripping whitespace, replacing specific characters (e.g., "*", digits) with their corresponding text representations, removing certain characters (e.g., "[]", ""̈), and replacing specific substrings (e.g., "Action ", "Info ") with empty strings or alternative representations.

4. **Pre-tokenization**: The text is pre-tokenized using a regular expression-based splitter that splits on whitespace characters.

5. **Post-processing and Decoding**: The `ByteLevel` post-processor and decoder are used to handle byte-level processing without adding prefix spaces.

6. **Padding**: The tokenizer is configured with padding parameters to pad sequences to the batch's longest length, with a maximum padding length corresponding to the context window size. The padding is applied on the left side using the `[PAD]` token.

7. **Training and Saving**: The tokenizer is trained on the generated transcript file using the configured trainer. The trained tokenizer is then saved to a JSON file with the specified output prefix.

Based on the provided information, here are some additional details that can be included in the appendix section to aid in reproducibility and provide academically important details:

Appendix B. Model Architecture and Training

B.1. Model Configurations

We experimented with several language model architectures, including LLAMA, RWKV, RecurrentGemma, and Mamba. However, due to time constraints, we focused on training and evaluating the LLAMA models. The specific configurations for the LLAMA models are as follows:

- Number of attention heads: 2
- Number of key-value heads: 2
- Number of hidden layers: 15
- Hidden size: 512
- Intermediate size: 512
- Tied word embeddings
- RMS norm epsilon: 1e-5
- ROPE theta: 500000

B.2. Training Setup

The key training parameters are:

- Learning rate: 1e-4
- Batch size: 8
- Evaluation interval: Every 1500 batches
- Context window sizes: 2048, 4096, and 8192

Appendix C. Data and Code Availability

The synthetic Blackjack dataset generator, along with the training and evaluation datasets, are publicly available on GitHub:

- Data Generator: `https://github.com/DhruvDh/llm-blackjack-synthetic-language`
- Training and Evaluation Datasets:
  `https://github.com/DhruvDh/slm-blackjack-synthetic-training`

The code for the model architecture, training script, and evaluation is provided in the appendix and can be used to reproduce the results presented in this report.