Team 1 ARC Challenge

1 OVERVIEW

Group members (first name, last name, student ID)

FIRST NAME	LAST NAME	STUDENT ID
Choekyel	Nyungmartsang	21-876-693
Joel	Kessler	21-875-232
Flavio	Kluser	21-874-854
Fabien	Morgan	21-876-727

Executive summary (max. 2000 characters, description of task, summary of method and results)

In this project, which was done as part of the HSLU NLP course, the goal is to develop solutions for the AI2 Reasoning Challenge (ARC). The challenge consists out of a dataset with grade-school level multiple-choice questions with the objective of achieving the maximum accuracy. The project is divided into multiple tasks to allow incremental progress from initial data analysis over modeling, to evaluation and error analysis. First it was decided to do a thorough data analysis to establish a good understanding of the data and to pave the way for subsequent tasks like preprocessing. Only the arc-easy dataset has been used for modeling. To examine meaning and context, topic modeling was done. The first model consists of a custom neural network classifier with different word embeddings. After setting a baseline, a randomly initialized BERT model, which is based on the Transformer Encoder Architecture, was trained. Subsequently, a pretrained BERT model was finetuned on the ARC dataset. All models have been optimized with Optuna to find ideal hyperparameter settings and to create a meaningful comparison. The evaluation has been done with a sample of 80 questions in the arc-easy set, which have been evaluated on various LLMs (Llama-2 70B, Bard, …). This showcases the drastic accuracy increase with LLM models. To wrap this project up, an error analysis was done to investigate the inner workings of our models with saliency maps and confusion matrices and spot potential for improvement in our model design. Additionally, to the base requirements, further topics were covered: Hyperparameter Tuning, Training on NLI datasets with different topics, Pretraining on biomedical data and Ensembling.

2 MOTIVATION

2.1. What prior work did you find? Explain briefly how each method works.

arXiv:1803.05457 [cs.AI] (https://arxiv.org/abs/1803.05457)
- Overview of the challenge and the content of the data.
- Results and reviews of different models, such as baseline performance, ARC corpus and infographics.

AI2 Reasoning Challenge (ARC) 2018 (https://allenai.org/data/arc)
- Offers a wonderful overview of the content of the data and shows the complexity of the reasoning tasks involved
- Provides the dataset and includes how other participants approached solving these challenges.

BertForMultipleChoice
(https://huggingface.co/docs/transformers/v4.28.1/en/model_doc/bert#transformers.BertForMultipleChoice)
- Bidirectional Encoder Representations from Transformers (from now on BERT) model designed for multiple-choice question answering.
- Example section shows how the question-and-answer choices should be tokenized.

3 TASK

3.1 Describe the task. What should be predicted? From what? What are the options?

The task is to solve multiple-choice reasoning questions and test how comprehensive and logical the inference is, given a set of multiple-choice question. The dataset is provided by the Allen institute for ai and is a worldwide renowned challenge. The goal of the task is to correctly predict the answer from the provided options for each given question. The question's level of difficulty spans grades 3 to 9, offering between 3 to 5 choices, with fewer than 1% featuring anything other than 4 choices. So, we filtered those out in our training and evaluation, because it would not impact anything.
The whole challenge is also split up in two levels, 'easy' level, and 'challenge' level.

Example of an 'easy' level multiple-choice reasoning question:
Which factor will most likely cause a person to develop a fever?
Answer choices:
[X] a leg muscle relaxing after exercise
[✓] a bacterial population in the bloodstream
[X] several viral particles on the skin
[X] carbohydrates being digested in the stomach.

4 DATA

4.1. Give a high-level summary of the dataset, including some dataset statistics.

Partitioning: 2,590 'challenge' questions, 5,197 'easy' questions and each set further split into Train, Validation and Test subsets.
The following table shows the distribution of questions across subsets.

Challenge Set	Questions	Easy Set	Questions
Train	1,119	Train	2,251
Validation	299	Validation	570
Test	1,172	Test	2,376

Language: English language on different grade levels ranging from 3 to 9.
Questions: 7787 grade-school level multiple-choice science questions (not a retrieval-based algorithm).
The following table shows the minimum and maximum lengths of the encoded tokenized questions.

Challenge Set	Min Length	Max Length	Easy Set	Min Length	Max Length
Train	7	128	Train	7	117
Validation	7	123	Validation	7	94
Test	6	157	Test	6	144

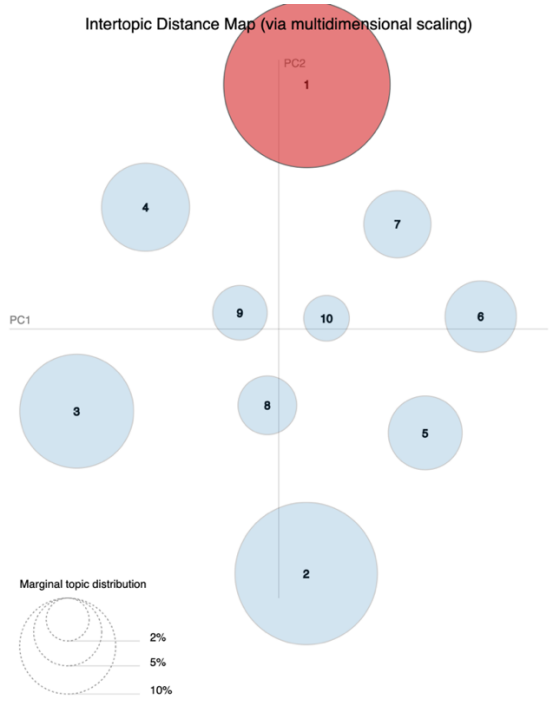
Answers: Each question has between 3 and 5 choices each, where 99% of them have 4 choices
The following table shows the minimum and maximum lengths of the encoded tokenized answers.

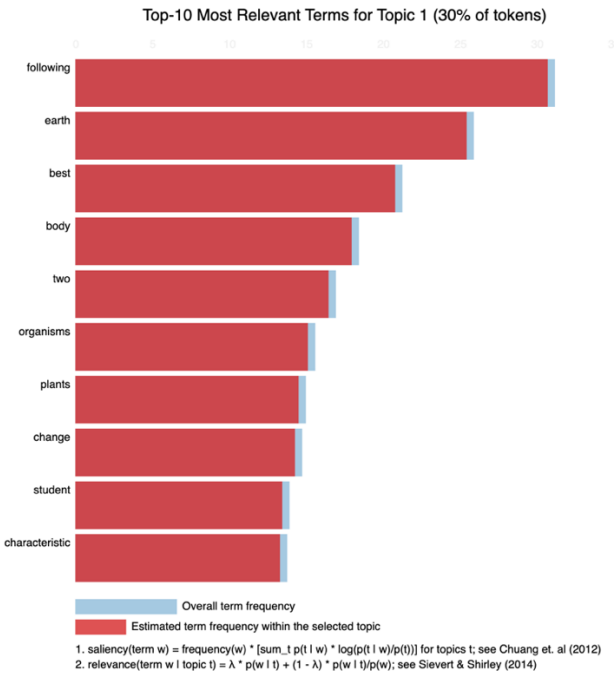
Challenge Set	Min Length	Max Length	Easy Set	Min Length	Max Length
Train	3	42	Train	3	49
Validation	3	36	Validation	3	20
Test	3	49	Test	3	35

Metric: Accuracy
Columns: Unique question identifiers (id), Main question text (question), Array of answer options with text and labels (choices), Correct answer key (answerKey)

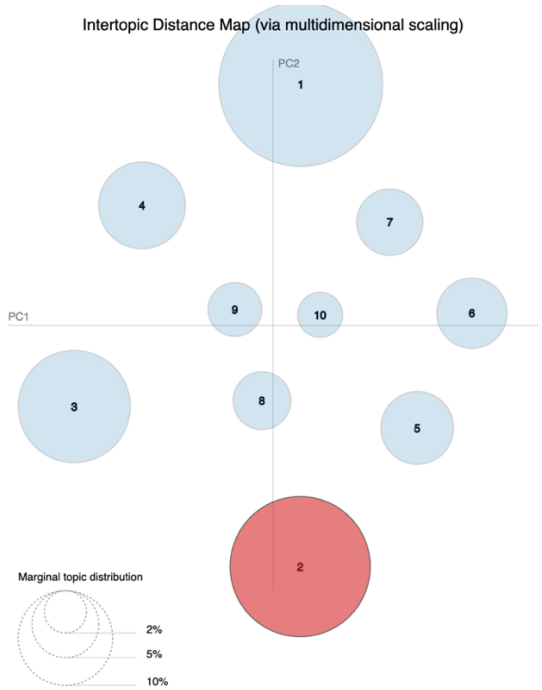
4.2. Describe the further results of your data analysis.

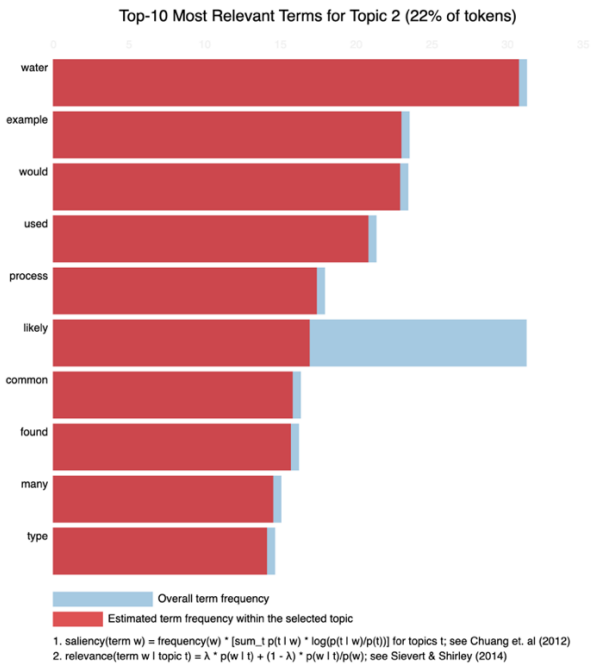
We used topic modeling to analyze if any important theme or subjects appear in the question dataset. So, we can get an overview of the questions, without looking at each question individually. The following preprocess was used: removing stop words, bi-grams and tri-grams.





The first topic of 30% tokens revolves around study of earth and living organisms. This is a good indication and validates the information, that the dataset contains science questions.



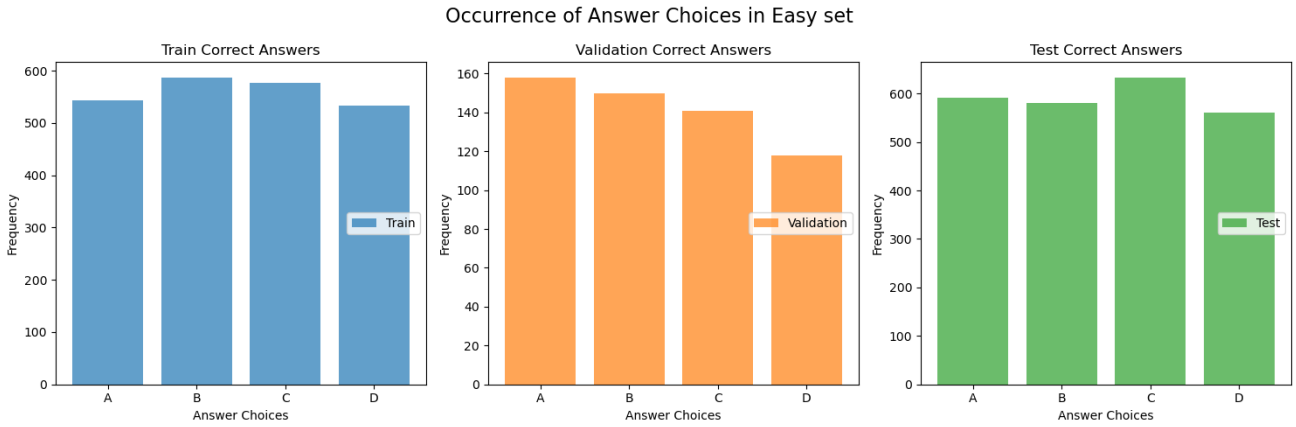


The second topic of 22% tokens revolves around the properties and occurrences of water.

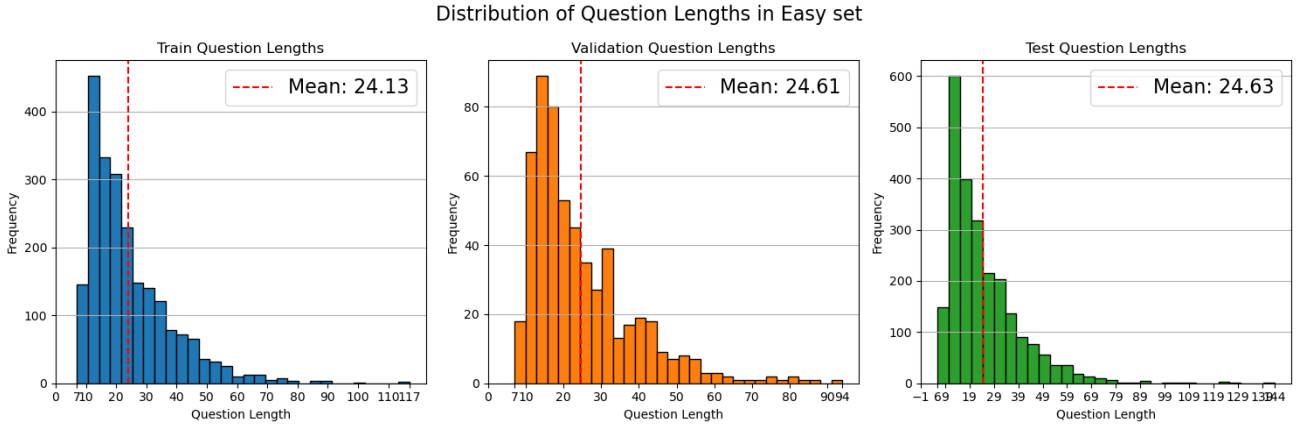
With this insight, we got a glimpse into the themes and subjects within the questions, showing the diversity of topics covered in the ‘easy’ dataset. This also indicates that the dataset is restricted to a specific domain of science, which could affect the generalizability of models trained on this dataset.

4.3. What are potential problems with the dataset? Is the data balanced? Does/could it contain biases?

There isn't a major imbalance between the answer choices in a specific set, but the 'challenge' set has fewer question examples than the 'easy' set, which could impact model training and evaluation. But we only focus on the 'easy' set so we do not have that problem.
The following image shows the occurrence of answer choices in the 'easy' set (without considering option 'E').



This image shows clearly that there is no bias in the answer option choices because that could influence model learning in which a particular position could be prioritized.
Some questions might lack sufficient context, so we plotted the distribution of question lengths in the 'easy' set. With most of the question having a length of 20 we can assume the questions are challenging even for humans to answer correctly. (Ton de Jong, https://doi.org/10.1007/s11251-009-9110-0)



4.4. Did you mitigate any of the problems mentioned in 4.3? If yes, how? If no, how would you do it?

To balance the distribution in 'easy' and 'challenge' set, we could use strategies like data augmentation, resampling techniques (oversampling or under sampling). But since we only focused on easy set it wasn't necessary.
To prevent the model of learning position specific answers, we could use shuffling to mix the order of the answer options across questions during training.
Regarding the insufficient context in some questions with small length, we could add context or provide supplementary information where needed. That could improve the datasets quality. (But hard to do in our opinion).
For longer questions, we could simplify the questions with segmentation or restructuring them to get a better understanding of the question.

5 PREPROCESSING

5.1. Which preprocessing steps did you apply? Why did you select these steps?

We preprocessed all questions in 4 steps.

1. At first, we decided on which dataset to use. The task given defined that we only needed to train on the 'easy' level, so we only focused on this set. The train, test and validation split were already given by the dataloader.
2. Because less than 1% do not have 4 answers, we decided on removing those questions from the dataset. This reduced the complexity of the project by a lot and helped us put more resources in creating a transformer that solves more than 99% of the questions as well as possible.
3. We mapped the correct answer (identified by 'answerKey') to a label (0, 1, 2, or 3) corresponding to the positions of the multiple-choice options and create labels for each choice indicating whether it's the correct answer.
4. For the word embedding we used word tokenizer which split every word into its own token. In the transformer we used the tokenizer and encoding from Huggingface to tokenize the question and its answer choices, so it is compatible with Transformer models.
5. In the word embedding, the average of all embeddings was taken as one vector to have a constant size. For the transformer, we truncated and padded sequences to a maximum length, which ensures uniformity for the model's input. The maximum length is calculated by the sum of the longest tokenized question, the longest tokenized answer choice and the three special tokens [CLS], 2x[SEP].

The steps for the transformer were selected because we looked at the Huggingface documentation and they provided an example for multiple choice question answering. (https://huggingface.co/docs/transformers/v4.28.1/en/model_doc/bert#transformers.BertForMultipleChoice).

6 MODEL

6.1. Briefly describe the models you experimented with. Why did you try these models, i.e. why did you expect they will perform well on your task?

Word embeddings: Before we started training our own models, we decided on making a baseline measurement with a simple word embedding to see how well our models perform compared to the simplest implementation, where a model tries to predict the answer just based on the words that appear in the question. This model just tries to guess the answer based on the meaning of the words in the question and the possible answers. The model doesn't try to understand the question. We looked at the three most popular models (Word2Vec, Glove and FastText) and removed stop words in the embeddings. To use the word embedding to predict the correct answer, we used a neural network classifier that takes 5 embeddings (Question, Answer 1/2/3/4) as input. These input embeddings would be the average embedding of all word embeddings of the question and the four answers. The return value of the network is a probability of 4 output neurons, which represent each possible answer.

Randomly initialized Transformer: We used BertForMultipleChoice initialized by BertConfig from Hugging Face Transformers. *["Initializing with a config file does not load the weights associated with the model, only the configuration."]* (https://huggingface.co/docs/transformers/v4.28.1/en/model_doc/bert#transformers.BertForMultipleChoice) and this default configuration will be similar to that of 'bert-base-uncased' *["Instantiating a configuration with the defaults will yield a similar configuration to that of the BERT bert-base-uncased architecture."]* Which we used for the pretrained model. This lets us compare the pretrained with the randomly initialized model.

Pretrained Transformer: We used AutoModelForMultipleChoice ('bert-base-uncased') also from the Hugging Face Transformers library. This pretrained model uses the BERT architecture and is trained on a large corpus.

We chose the randomly initialized model after choosing the pretrained model because we wanted to compare the performance from the same model's name. We expected the randomly initialized model to perform comparatively lower than a pretrained model because that model has no knowledge or is not fine-tuned on a specific task. In the other hand, a pretrained model has been trained on a large corpus of text data and should give a better performance. In addition, fine-tuning on our task should also benefit the performance.
The reason we used 'bert-base-uncased' is that BERT in general has a strong performance across different NLP tasks, (arXiv:1810.04805v2 [cs.CL] Table:1) because it captures the contextual information very effectively. We also expect it to encode the questions and answer choices into meaningful representations and give accurate predictions using its knowledge from the large corpus it was pretrained on.

Ensemble: We explored and experimented with a range of models and used the BERT ('bert-base-uncased') and the BioBERT ('biobert-base-cased-v1.1-squad') model to make a robust final prediction. We combined the two models' predictions by taking the average of the predictions. This would use the strength of each model and compensate for individual model shortcomings.

6.2. What is your input and answer format? Show an example.

Data	View
Question	'Which factor will most likely cause a person to develop a fever?'
Answer	'a leg muscle relaxing after exercise' 'a bacterial population in the bloodstream' 'several viral particles on the skin' 'carbohydrates being digested in the stomach'
input_ids 101 = [CLS] 102 = [SEP] Pad = 30* **just for example*	[[101, 2029, 5387, 2097, 2087, 3497, 3426, 1037, 2711, 2000, 4503, 1037, 9016, 1029, 102, 1037, 4190, 6740, 19613, 2044, 6912, 102, 0, 0, 0, 0, 0, 0, 0, 0], [101, 2029, 5387, 2097, 2087, 3497, 3426, 1037, 2711, 2000, 4503, 1037, 9016, 1029, 102, 1037, 17341, 2313, 1999, 1996, 2668, 21422, 102, 0, 0, 0, 0, 0, 0, 0], [101, 2029, 5387, 2097, 2087, 3497, 3426, 1037, 2711, 2000, 4503, 1037, 9016, 1029, 102, 2195, 13434, 9309, 2006, 1996, 3096, 102, 0, 0, 0, 0, 0, 0, 0, 0], [101, 2029, 5387, 2097, 2087, 3497, 3426, 1037, 2711, 2000, 4503, 1037, 9016, 1029, 102, 2482, 5092, 10536, 7265, 4570, 2108, 17886, 2098, 1999, 1996, 4308, 102, 0, 0, 0]]
attention_mask	[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]]
labels	[0, 1, 0, 0]

This shows the tokenized inputs of one question with the four different choices and the label indicating the correct question and choice pair.

6.3. What hyperparameters do your models have?

Word Embedding	**Bert-base-uncased**	**Biobert-base-cased-v1.1-squad**
Embedding: FastText Activation Function: Tanh Learning rate: 0.0001 Optimizer: Adam Stopwords: False Layers: 3 Hidden Dimensions: 4000	Batch: 16 Epochs: 6 Learning rate: 0.0001 Optimizer: AdamW (weight_decay=0.3) Scheduler: CyclicLR (base_lr=0.001, mode='triangular2') attention_probs_dropout_prob: 0.2 hidden_dropout_prob: 0.2 classifier_dropout: 0.2	Batch: 16 Epochs: 6 Learning rate: 0.0001 Optimizer: AdamW (weight_decay=0.3) Scheduler: CyclicLR (base_lr=0.001, mode='triangular2') attention_probs_dropout_prob: 0.1 hidden_dropout_prob: 0.15 classifier_dropout: 0.05

Since the pretrained models were overfitting on the training data we focused on weight decay and dropouts on both BERT models.

7 TRAINING

7.1. Describe your training methodology. How did you tune the model's hyperparameters? How did you select the best model?

Embedding: For the embedding, we used a hyperparameter optimizer. We set up the data and network in a way, where the hyperparameter optimizer could optimize the following factors:

- Hidden layers (1–6)
- Hidden dimensions (400 – 5'000)
- Remove stop words (True or False)
- Embedding type (word2vec, glove, fastText)
- Optimizer (SGD, Adam)
- Activation function (ReLU, Sigmoid, tanh)
- Learning rate (1e-1, 1e-4)

For the hyperparameter optimizer we choose Optuna which figures out the best hyperparameters by trying to maximize or minimize a target value and for every experiment calculates the potentially best hyperparameters based on previous results. As a target, we choose to minimize the top 10% of the validation loss. The reason for this specific measurement and not just the lowest validation loss is that we noticed in previous runs, that out of coincidence there were runs with a high number of layers, hidden dimensions and learning rate, that for one epoch reached a low validation loss, but that score didn't represent it's true performance. Because optuna tunes hyperparameters based on how well the previous runs where, this meant that it started to focus on models that didn't perform well. The model that got saved was always the one with the lowest validation loss and not the last run in the training.

Finetuned Transformer: For tuning our hyperparameter in a pretrained model we manually explored the parameters rather than relying on a grid search or dedicated optimizer like Optuna. Initially, we identified some parameters that could be modified like: batch size, optimizer with weight decay, different regularization techniques, epochs, and specific BERT configurations like hidden_dropout_prob, classifier_dropout and attention_probs_dropout_prob. For learning rate, we stayed throughout the tuning on 0.0001, which we considered optimal for fine-tuning.
Significantly observing overfitting on the training dataset over larger epochs, we systematically changed one parameter at a time while closely monitoring convergence (train loss, validation loss, train accuracy and validation accuracy). This iterative process involved experimenting with various dropout rates and shorter epochs to get some knowledge on how accurate our model will be.

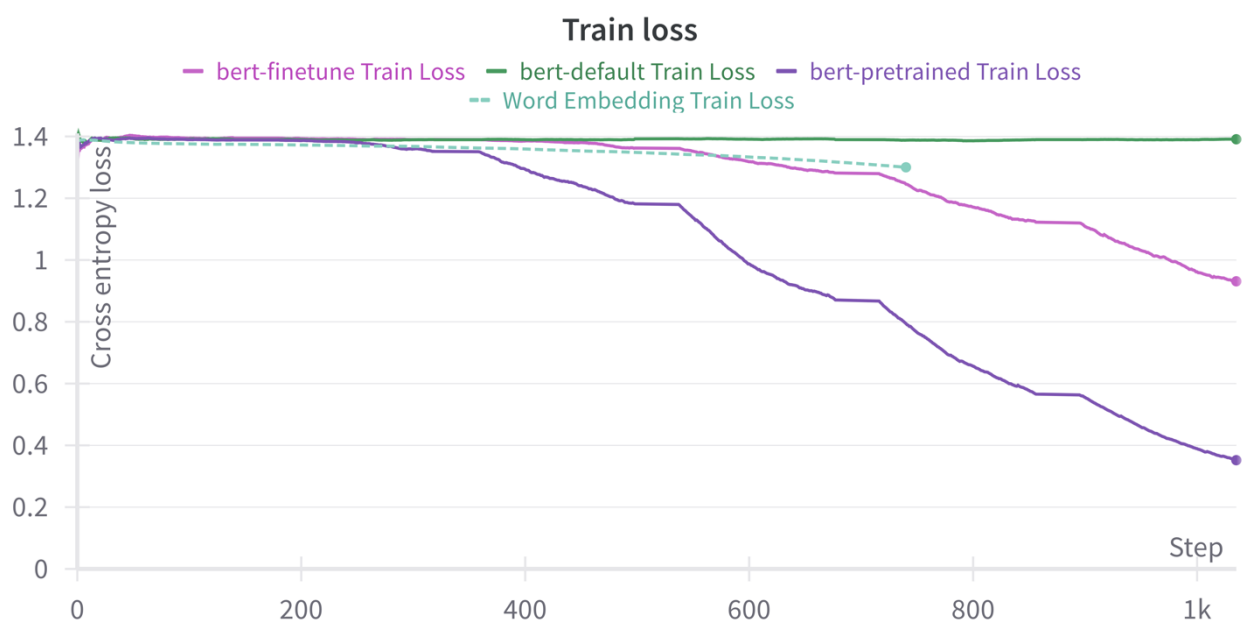
In the appendix, we included two tables outlining our hyperparameter adjustments for reference. To select the best-performing model, we focused on the validation loss, aiming for a consistent decrease throughout the epochs. The model showing the best validation loss became our primary consideration for comparison among others, leading us to identify the best model for further evaluation.

7.2. Which training hyperparameters had a big impact, which ones did not?

Word Embedding: The embedding itself was one of the biggest differences, and the decision to remove the stop words made a big impact. Having a high number of hidden layers and hidden dimensions only made the model to overfit to the training data.

Finetuned Transformer: Among the training hyperparameters, we focused on regularization techniques and dropout rates for handling overfitting. We experimented with different regularization methods like OneCycleLR and ReduceLROnPlateau, which should be effective in handling overfitting. Surprisingly, these techniques didn't significantly change the validation loss in our case. With that we decided on using CyclicLR in triangular2 mode. This cyclic learning rate aims to smoothly anneal the learning rate between two boundaries, helping the model navigate parameter space more effectively and escape local minima.
Epochs were held standard at 6, and adjustments were rare unless there were specific cases where the model indicated potential for enhanced performance over additional epochs.
One of the most impactful parameters was the manipulation of hidden_dropout_prob. This adjustment significantly affected the model's performance compared to other parameters we explored, demonstrating its notable influence on combating overfitting and improving the model's generalization.

7.3. Show a plot of your final model's training performance vs. the number of training steps/epochs. You can combine this with 8.2. if you like.



7.4. How long did training take for each final model you trained? Did you observe high variance within the same model?

Word Embedding: Training for 1'500 epochs took on GPUHub on average 30 minutes. Because we added some smart early stopping rules and most runs started overfitting after 100 epochs (200 steps) this meant that all runs where finished at the end after 3.5 hours.

Finetuned Transformer: Training, evaluating and finetuning a pretrained model took us around 37 minutes to an hour. Regarding high variance in the context of training we did not see any significant variability in model performance when trained multiple times on the same data with different random initializations or data shuffling. However, regarding high variance in the context of training time due to experimenting with different parameters was noticeable. But it wasn't a problem, as long as the training time converges, and the model's performance gets better.

7.5. How many experiments did you perform for each model? Could one of the models have an advantage from more training runs?

Word Embedding: In total, for word embedding, we trained about 150 models. These training runs got split up in about 10 runs at the beginning to get it running and try out a few hyperparameters. Then 70 runs on fastText with optuna hyperparameter tuning. Finally, a 70 trial run with additional hyperparameters as stop words removal and 2 more embedding options (Word2Vec, Glove). If we look at the hyperparameters of the best run there is only the number of layers, that are at the border of the hyperparameter value range. So maybe redoing the run with a higher range of layers could make the model better.

Finetuned Transformer: We did more than 20 experiments for each pre-trained model and after a while the improvements became marginal and did not significantly impact the model's overall performance. There could be potentially a better configuration of parameters which may improve the performance in more training runs but extensive hyperparameter tuning could also lead to overfitting to the validation set, making the model perform worse on unseen data.
More training runs could lead to a better model, which is why we had a hard time deciding on the "best model". At last, we decided on a model which is significantly better than a baseline or has a better validation loss than the pretrained model.

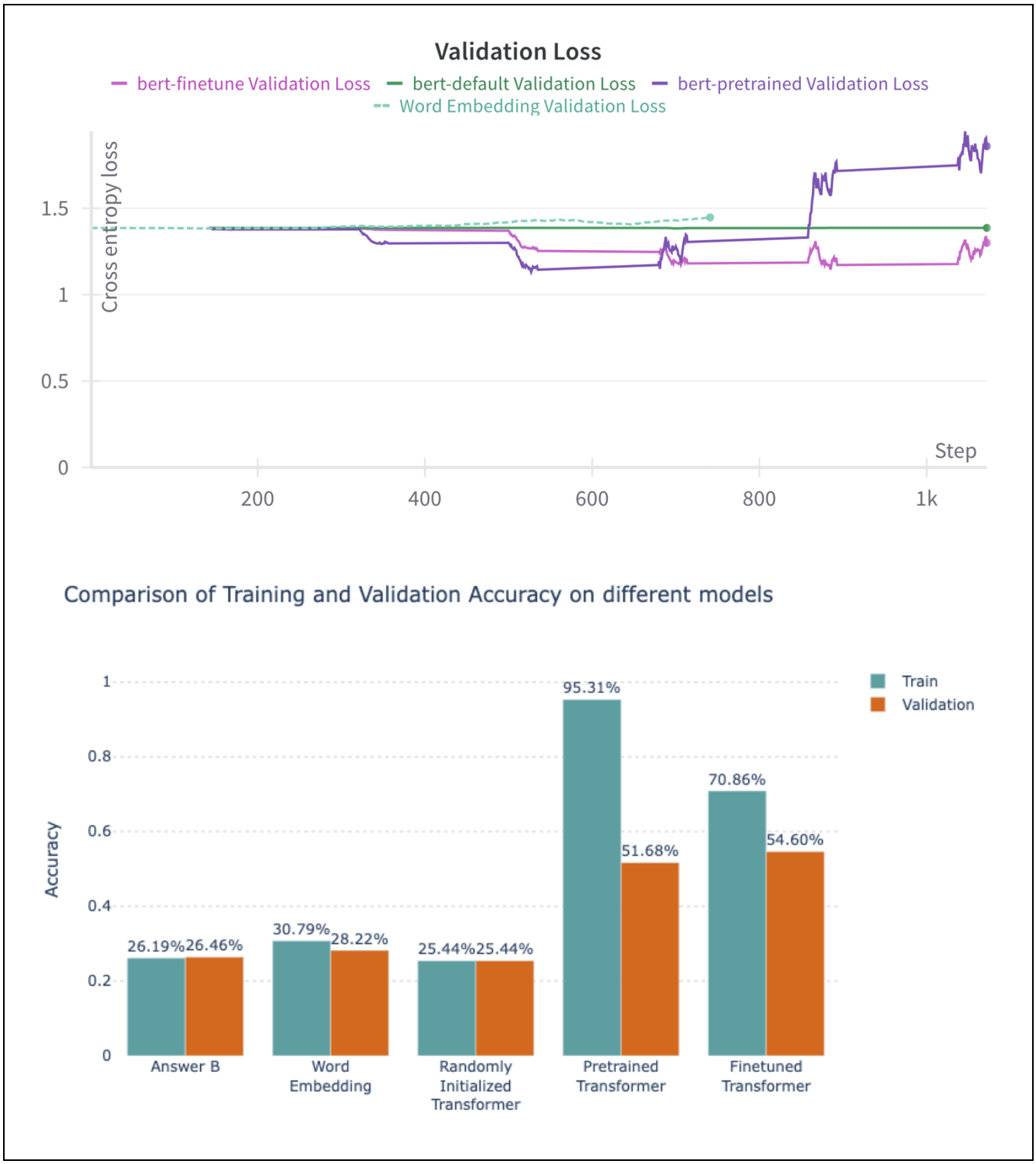
8 VALIDATION

8.1. Describe your validation setup. What metric did you choose? How often did you validate your model?

Metric: Accuracy
We used accuracy because this is the standard metric for classification tasks like multiple choice question answering. It indicates the correctly predicted answers out of the total.
Performance is evaluated at the end of each epoch, after training on the entire training dataset. This makes sure how the performance will be on unseen data and how it will generalize.

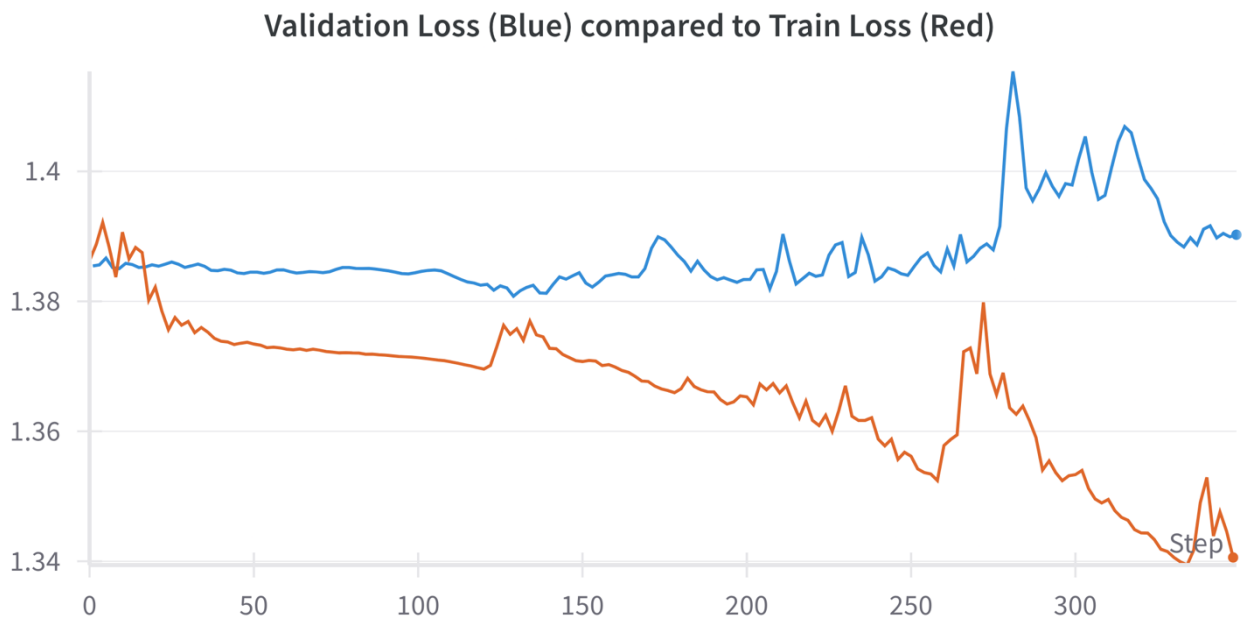
Our validation setup involves training the model using the training data and evaluation its performance on a separate validation set. The chosen metric for evaluation is accuracy, which measures the proportion of correctly predicted labels to the total number of predictions, shown as a percentage. The loss metric helps assess the convergence and stability of the model during training, while accuracy is a key performance indicator, showing the model's ability to make correct predictions on the validation set. The loss on the validation set is a key factor in this challenge and provides an insight into its generalization capability.
As for the frequency of validation, we monitor and asses at the end of each epoch the performance of the model. This helps us identify potential issues like overfitting or underfitting and helps us find the best performing model based on its performance on the validation set.

8.2. Show a plot of validation performance. Compare this to the training performance, if possible in the same plot.



8.3. Can you recognize an overfitting phase? Do you think your model should be trained for shorter/longer?

Word Embeddding: In the validation loss you can clearly see that from the start to the steps between 100 and 200 the validation loss goes down as expected but after that the validation loss gets out of hand and increases. Everything after that shows how the model starts to overfit.

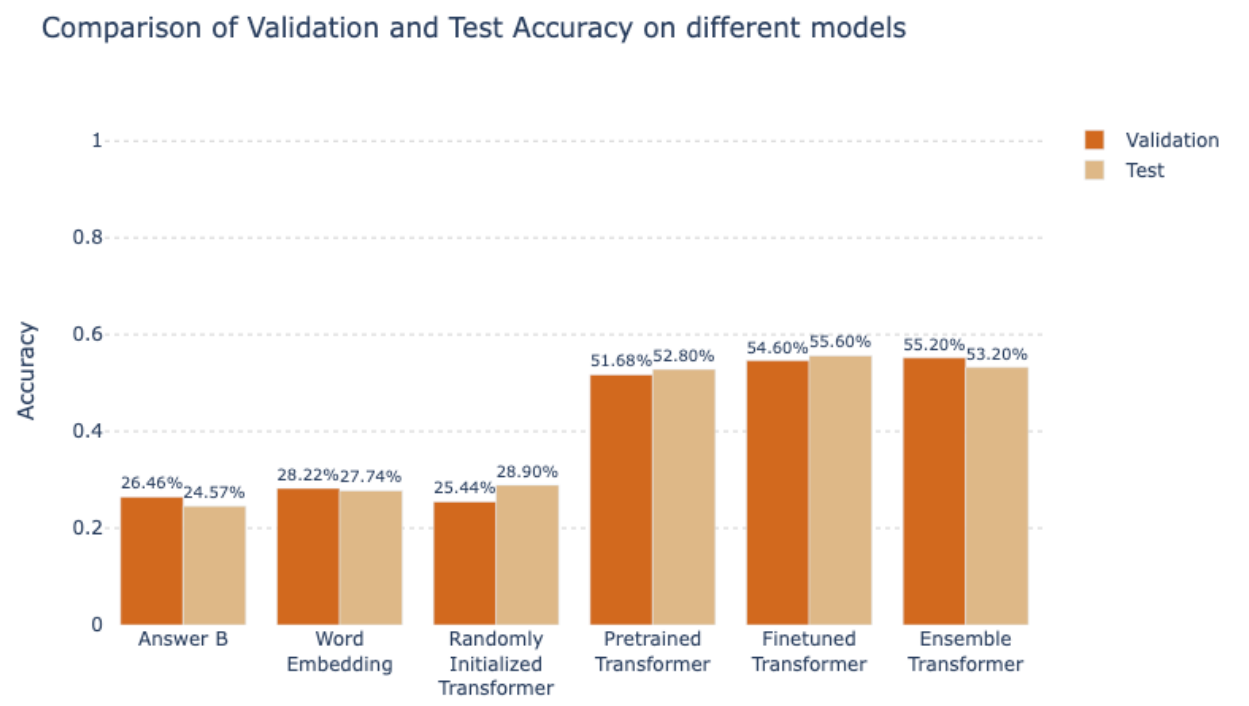


Finetuned Transformer: We also observed a big overfitting phase in our pretrained model. Noticeable first on the training set as the model started to learn specific details in the training data and then it also overfitted after certain epochs on the validation set. This shows us the limitation in generalization capability of the model. By using constraints like weight decay to penalize large weights, using dropout layers, and shortening the epochs showed a better result. At last, we balanced the epochs and regularization techniques to handle overfitting.

9 TESTING

9.1. How does your model's performance on the validation set compare to the performance on the test set? Is there a gap? If yes, can you speculate why?

On average the difference between the validation set and test set is minor as we can see in the image.

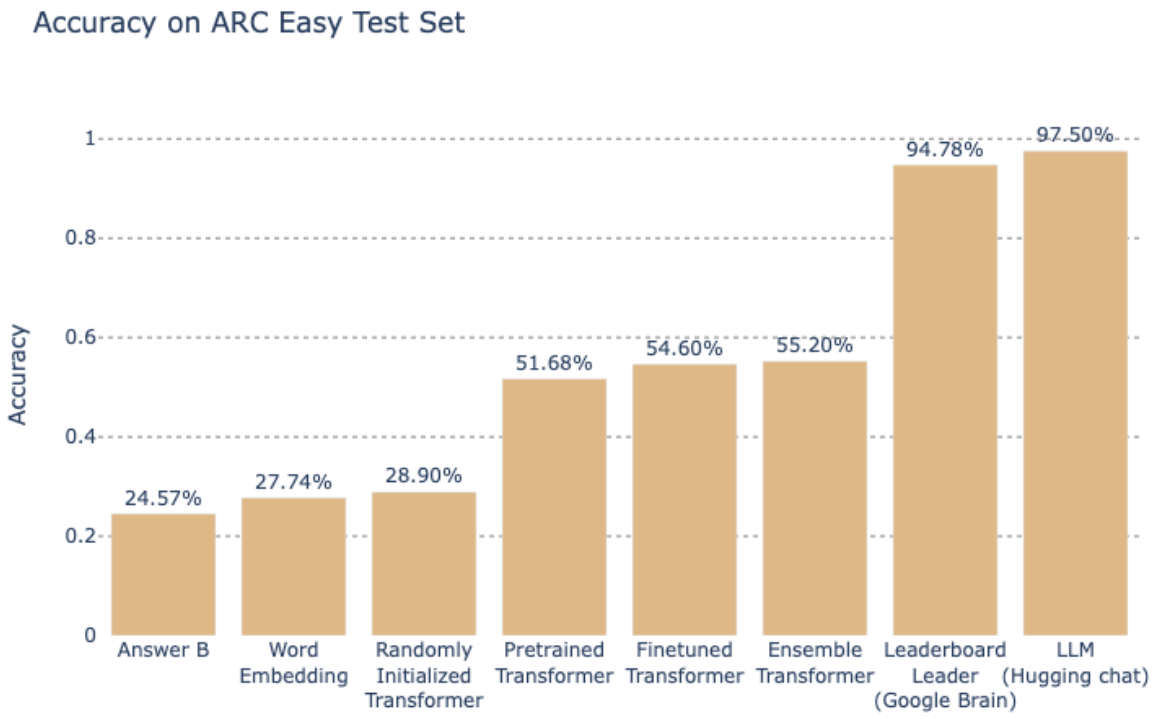


9.2. How many hyperparameter trials did you do for each model? Is there a possibility you have overfitted to the validation set?

Finetuned Transformer: We experimented with over more than 20 different hyperparameter for each model and checked the model's performance on a separate test dataset which was not used during training or validation. Most of the hyperparameter that we experimented with were regularization techniques so we would not overfit. We also always checked the performance on the test set and if there is a significant drop in that could indicate overfitting.
We have encountered mostly a similar performance on the validation and on the test set, so we did not use other strategies such as cross validation to mitigate an overfitting on the validation set. With cross validation it would split the dataset into multiple subsets for training and validation and the model would not rely on a single validation set.

10 EVALUATION

10.1. Compare the best model of each task. Which model performs best? Was this expected? Why?



"Answer B" Model: For our first benchmark, we looked at the lower bound and how well the most common answer in the training and validation set performs on the test set. The most common answer was B in these sets, so we looked at the accuracy of only answering questions with B on the test set.

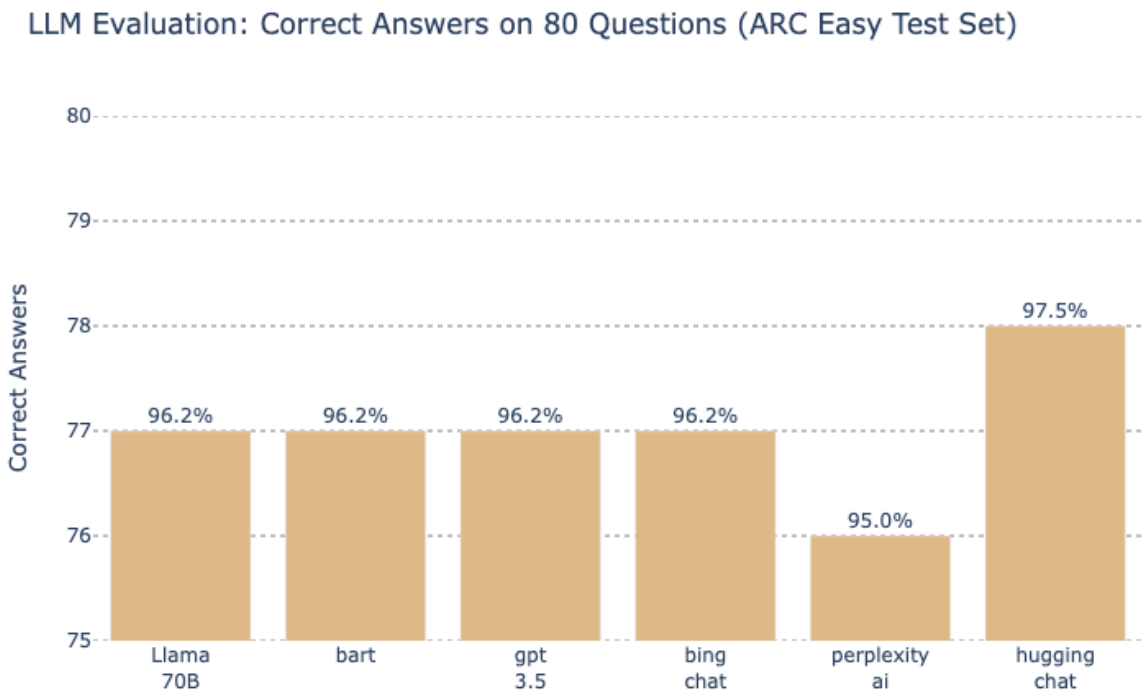
The result for our first experiment, where we calculated the accuracy of answering the whole test set with question B, was not surprising. We reached a score of "0.245" which is close to 0.25, the statistical probability that you answer a question correctly by just randomly guessing.

Embedding: To compare our transformer models with the current state of the art we also looked at two further resources. First, we looked at the ARC leaderboard to see how well we perform compared to the best scores that ever submitted their work and at the end we also compared these scores with the 6 most well-known LLMs on

a random set of 80 questions of the easy test set. We ran the 80 questions through Google Bard, Llama 70B, GPT 3.5, Bing Chat, Perplexity Ai, and Hugging Chat.

For the two other sources we anticipated that we wouldn't be able to beat them and that was also the outcome. The winner on the leaderboard currently is Google Brain with the Model ST-MoE-32B (State: 17.12.2023, Source: https://leaderboard.allenai.org/arc_easy/submissions/public) and an accuracy of 0.9478. Our pretrained model lost by quite a lot with a difference of 0.396 between ST-MoE-32B and our Ensemble Transformer model.

LLMs: The question if the current state of the art LLMs could beat that score was still interesting. The highest score was submitted in the year 2021 which meant that there weren't big advancements it that challenge for the last 2 years. At the same time, LLMs made a lot of progress in that time.
Interestingly enough, all LLMs performed very similar. The difference between the best and the worst performance of the 80 questions were 2 wrong answers. The winner was HuggingChat with 78 out of 80 questions correct.



This means HuggingChat reached an accuracy of 97.5%, which beats the current leaderboard spot. Looking through the responses, one question stood out. The question E2024 which goes the following:

"A small aluminum cube is dropped into a beaker of water to determine the buoyancy. Which of these is not necessary in determining the buoyant force acting on the cube?
(A) density of water
(B) displaced volume
(C) gravitational pull on Earth
(D) density of the aluminum cube"

This question was incorrectly answered by all LLMs. The correct answer would be "(D) density of the aluminum cube" but all the LLMs wrongly decided for "(C) gravitational pull on Earth". This question was interesting in multiple aspects. For example, we found that the question is quite complicated and were surprised to find it in the easy dataset. That all the chatbots decided to answer with the same wrong answer is also a surprise to us. The use of a lot of technical terms like beaker and buoyancy in the question stood out when we read the question as nonnative English speaker.

Finetuned Transformer: We also evaluated as bonus task our finetuned models on different multiple-choice datasets to see how our model's generalization ability is. We used the Measuring Massive Multitask Language Understanding in short MMLU dataset (https://huggingface.co/datasets/Stevross/mmlu) and chose three different tasks: college biology, high school biology and philosophy. We compared biology tasks with philosophy task to check how domain specific our model is. When we ran our ensemble models on the three tasks, we saw that our model performs better on the philosophy task rather than on the two biology tasks. We refrained from conducting extensive additional experiments, yet our reasoning leaned towards BERT potentially showing a stronger impact than BioBERT.

In conclusion, we can say that the order of the models in evaluation on the test set doesn't bring many surprises with it. The embedding is slightly better than just answer the question with the same answer, our transformers perform worse than the models on the leaderboard and the end of the chart still has a little surprise showing that LLMs beat the current leaderboard spots.

10.2. Do all the models make the same type of mistakes? If no, how are they different?

The reason for mistakes is heavily dependent on the model used. In the case of the Embedding model, it was observed that the model often relied on similarities between encoded question and answer. In practice this means, that it is more likely to predict the right answer if question and answer contain similar information. For the pretrained models we couldn't determine a specific, reoccurring pattern for mistakes. In the case of LLMs, we found that the model often preferred answers that sounded legit but didn't match since the question was altered in some way to make them seem unintuitive. Overall, the mistakes vary across the different models.

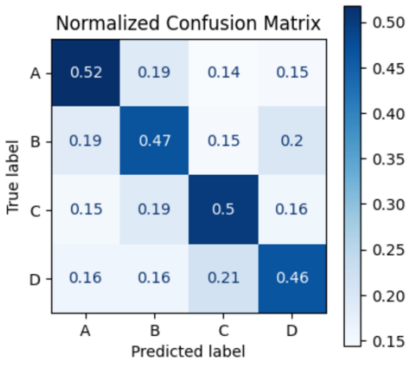
10.3. Do you think your results could change if you had more time and resources?

For our own models, accuracy could certainly be further improved by enhancing hyperparameter search or using larger models. For the LLM queries it is safe to assume, that accuracy could be improved by using prompt engineering. In the latter case however, there is little room for improvement since the baseline accuracy is already above 95% in our LLM evaluation.
We exclusively trained using encoder models, but there was potential to include decoder models, which might have further improved our performance. Also expanding the context by incorporating additional datasets or using the ARC corpus could influence the model's performance. If we compare our results against the ARC benchmark, it shows that there is potential for better results. Given more time and resources we would have gotten an improved performance.

11 ERROR ANALYSIS

11.1. What types of errors does your model make? Have you found a setting (knowledge type, reasoning type) where the model makes more errors than usual?

We analyzed the confusion matrix, and it shows that our model achieves approximately 50% accuracy along the diagonal, indicating that it correctly identifies the correct answer choice within the four option choices. However, there are some fluctuations and occasional deviations from the correct choice.



For how the model's decision-making process looks like, we used the Captum library to visualize where BERT focuses its attention on. Through this, we observed where the model placed more focus on specific tokens or segments within the question-and-answer choices. For instance, in certain scenarios, tokens crucial to the correct answer were highlighted in green, showcasing the model's focus.

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
20	20 (0.07)	-1	2.47	[CLS] sk ##unk ##s spray a strong - smelling liquid to _ _ _ . [SP] get rid of waste ##s [SEP] [PAD] [PAD]

As for identifying specific settings where the model makes more errors, we inspected questions and the four answer choices individually and looked at the correct answer in comparison to the predicted answer. We found that certain knowledge types or reasoning patterns were challenging for the model. For example, questions that required detailed scientific reasoning or those involving complex contextual understanding. This indicates the need for further experiments in handling such domain knowledge.

11.2. Inspect some of the examples where your model makes an error. Show at least one here. Can you imagine why it makes those errors?

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
19	19 (0.04)	12	0.94	[CLS] a small aluminum cube is dropped into a beak ##er of water to determine the bu ##oya ##ncy ,which of these is not necessary in determining the bu ##oya ##nt force acting on the cube ? [SEP] density of water [SEP] [PAD] [PAD]

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
19	19 (0.03)	40	0.91	[CLS] a small aluminum cube is dropped into a beak ##er of water to determine the bu ##oya ##ncy ,which of these is not necessary in determining the bu ##oya ##nt force acting on the cube ? [SEP] displaced volume [SEP] [PAD] [PAD] [PAD]

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
19	19 (0.03)	42	0.93	[CLS] a small aluminum cube is dropped into a beak ##er of water to determine the bu ##oya ##ncy ,which of these is not necessary in determining the bu ##oya ##nt force acting on the cube ? [SEP] gravitational pull on earth [SEP] [PAD]

Legend: Negative Neutral Positive

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
19	19 (0.04)	4	1.04	[CLS] a small aluminum cube is dropped into a beak ##er of water to determine the bu ##oya ##ncy ,which of these is not necessary in determining the bu ##oya ##nt force acting on the cube ? [SEP] density of the aluminum cube [SEP]

We used the Captum library to see how the model's decision-making process looks like. The given question above is about the buoyant force acting on an aluminum cube submerged in water. Interestingly, the model correctly identifies the concept of `gravitational pull on earth` and highlights these terms, indicating a strong focus on this aspect within the context. However, this focus on gravitational pull, the model predicts an answer that doesn't align with the target. The correct answer is the last option, where the model shows minimal attention or focus. This error between the model's attention and the correct answer raises interesting insights into the reasoning patterns or knowledge representation the model employs. We think that the model's overemphasis on gravitational pull might overshadow the significance of other critical factors, such as the cube's properties or the water's influence on buoyancy, leading to an incorrect prediction. This highlights a potential limitation in the model's contextual understanding or reasoning, where it fixates on certain aspects but fails to comprehensive understand the entire question, ultimately resulting in an incorrect answer.

11.3. While it is very difficult to pinpoint why exactly a model makes certain errors, can you speculate on potential shortcomings of your model that make it fail systematically on certain inputs?

We think that the model struggles with nuanced contextual comprehension and focuses on specific keywords or phrases without fully understanding the contextual relevance or relationship with the whole question. Also, the domain specific knowledge is very important and choosing the correct pretrained model with the right depth or specificity of science knowledge is challenging. These are some of the shortcomings why our model fails on certain inputs.

12 IMPROVEMENTS

12.1. What would be your next steps to improve your model?

Expanding the dataset with more diverse and challenging questions could cover a wider topic and question types. Testing other pretrained models and adjusting hyperparameters of those models could also improve our final model's performance. Also adding the ARC corpus to incorporate further information and maybe taking a better look at the attention mechanisms which captures the weight important context could also benefit to improve our final model.

12.2. What other models would also be interesting to run if you had more time/computational resources?

For the models we have developed, accuracy could certainly be further improved by enhancing hyperparameter search or using larger models. But hyperparameter tuning and longer training sessions can only do so much. If there wouldn't be a time limit neither a resource limit, we would train bigger pretrained model like Llama and train it not only on the science questions from the training set but furthermore on other questions. This could increase the accuracy of llama even more than we have seen on the LLM benchmark. Other model variants of BERT and maybe incorporating decoder models would be interesting to run.

12.3. What additional data would most benefit your current model's performance on this task?

Adding the ARC corpus for better context and introducing a variety of question formats (not just multiple choice) could train the model to handle diverse question types and make the model better on comprehension.

13 PROJECT INFORMATION

13.1. List all the tools you used in this project, and what you used them for. This includes AI tools for coding/writing assistance.

Github Copilot/Chat GPT:	Accelerate coding development, used for word embeddings and visualizations.
Llama 2:	To complete task 4: Evaluate an LLM. Link: https://www.llama2.ai/
Bing Chat:	To complete task 4: Evaluate an LLM. Link: https://www.bing.com/
Chat GPT:	To complete task 4: Evaluate an LLM. Link: https://chat.openai.com/
Perpexity:	To complete task 4: Evaluate an LLM. Link: https://www.perplexity.ai/
Hugging Chat:	To complete task 4: Evaluate an LLM. Link: https://huggingface.co/chat/
Bard:	To complete task 4: Evaluate an LLM. Link: https://bard.google.com/
Microsoft Word:	Documentation of results in NLP Canvas
Microsoft PowerPoint:	Final presentation
GitHub:	Hosting of all project files

13.2. List all your team members and what they did.

Choekyel Nyungmartsang:	Task 0: Data Analysis Task 2: Train a randomly initialized Transformer, Task 3: Finetune a pretrained model
Joel Kessler:	Task 2: Train a randomly initialized Transformer, Task 3: Finetune a pretrained model
Flavio Kluser:	Task 4: Evaluate an LLM Task 5: Error Analysis
Fabien Morgan:	Task 1: Train a word embedding baseline Task 4: Evaluate an LLM

Appendix

bert-base-uncased

Batch	Optimizer (AdamW)	Regularization (CyclicLR)	Epochs	Train accuracy	Validation accuracy	**Note:** **w**=weight_decay, **h**=hidden_dopout_prob, **a**=attention_probs_dropout_prob, **c**=classifier_dropout
16	w=0.1	mode=triangular2	6	25.44	28.92	Bert randomly initialized
16	w=0.1	mode=triangular2	6	95.31	51.68	Bert pre-trained, overfit on training set
16	w=0.1	**ReduceLROn-Plateau**	5	24.68	32.63	h=0.2, a=0.2, loss stays same
16	w=**0.3**	**OneCycleLR**	4	24.50	27.87	h=0.2, a=0.2, loss stays same
16	w=**0.1**	mode=triangular2	6	82.55	52.91	h=0.2, a=0.2, overfit on training set
16	w=**0.3**	mode=triangular2	6	80.5	51.50	h=0.2, a=0.2, overfit on training set
16	w=0.3	mode=triangular2	6	85.90	57.50	**h=0.2**, overfit on training set
16	w=0.3	mode=triangular2	6	61.13	49.03	**h=0.3**, better result
16	w=0.3	mode=triangular2	**12**	92.19	47.8	h=0.3, overfit on training set and bad result
16	w=0.3	mode=triangular2	6	86.57	54.67	**a=0.2**, overfit on training set
16	w=0.3	mode=triangular2	**8**	80.28	51.32	**h=0.2**, **a=0.2**, **c=0.05**, overfit on training set
16	w=0.3	mode=triangular2	6	75.43	51.24	h=0.2, **a=0.3**, overfit on training set
16	w=0.3	mode=triangular2	6	62.52	51.32	**h=0.3**, a=0.2, better result
16	w=0.3	mode=triangular2	6	68.94	49.21	h=0.3, **c=0.1**, good result

1

biobert-base-cased-v1.1-squad

Batch	Optimizer (AdamW)	Regularization (CyclicLR)	Epochs	Train accuracy	Validation accuracy	**Note:** **w**=weight_decay, **h**=hidden_dopout_prob, **a**=attention_probs_dropout_prob, **c**=classifier_dropout
16	w=0.1	mode=triangular2	6			Bio Bert randomly initialized
16	w=0.1	mode=triangular2	6	93.44	53.97	Bio Bert pre-trained, overfit on training set
16	w=0.1	mode=triangular2	6	62.96	49.38	**c=0.1**
16	w=0.1	mode=triangular2	6	41.01	51.15	**h=0.2**
16	w=0.1	mode=triangular2	6	79.16	48.68	**h=0.2, a=0.2,** starting to overfit
16	**w=0.3**	mode=triangular2	6	78.80	52.38	h=0.2, a=0.2, overfit on training set
16	w=0.3	mode=triangular2	6	85.23	55.56	h=0.2, **c=0.1**, overfit on training set
16	w=0.3	mode=triangular2	6	56.22	52.20	**a=0.2**, c=0.1, better result
16	w=0.3	mode=triangular2	6	64.03	53.62	**c=0.1,** good result
16	w=0.3	mode=triangular2	6	78.40	50.26	**h=0.2**, **a=0.2**, c=0.1, overfit on training set

2