Classification Approach

	w/ zero-shot	w/ fine tuning
Validation Accuracy	24.8%	57.2%
Testing Accuracy	25.2%	57.6%

Data Preprocessing

For each row in the jsonl file and for 0 <= i <= 3, we construct training examples as below:

Input: row[fact1] + ‘ [SEP] ‘ + row[question][stem] + row[question][choices][i][text]
label: 0 or 1

An input example is constructed by concatenating “fact1” with the question stem and a particular choice for the question (as shown in pseudocode above). The label assigned to the input would then be “0” if the choice is incorrect and “1” if the choice is correct. The input examples are then shuffled and passed into the model with batch size 100.

Model

We use the pretrained “bert-base-uncased” model from HuggingFace as a base model, and we used the “bert-base-uncased” tokenizer to determine the model parameters. We additionally adjust the hidden and attention dropout probabilities using AutoConfig from HuggingFace. The training examples are first passed directly into the pretrained model. We then pass the output from the pretrained model into a fully connected linear layer (maps from 768 inputs to 1 output), the output from that then gets mapped to a sigmoid layer, which outputs a probability between 0 and 1.

Hyperparameters

For the BertTokenizer, we used return_tensors=’pt’, padding=’max_length’, max_length=128. For the config of the pretrained model, we have hidden_dropout_prob=0.5 and hidden_dropout_prob = 0.5. We used the Adam optimizer for the model’s parameters with learning rate=0.0001. We used a binary cross entropy as the loss function. We trained on a batch size of 100. We also used an exponential learning rate scheduler to adjust learning rate with gamma=0.1. The model is trained for 4 epochs. Sequence max length is 256.

Training loop

Standard training loop for batch training is used, where the training loss for each batch is computed and used for backpropagation to update the parameters of the model through the optimizer. The entire training dataset is trained in batches at every time step. Training loss and validation loss is computed and stored at each epoch. The accuracy is computed using classification report from sklearn by passing in predicted labels from the test and validation sets.

Zeroshot

To do zeroshot training and inference, we mark all the parameters in the model as requires_grad=False, such that they will not be included in the computation of gradients during backpropagation. Everything else remains the same.

Generative Approach

	w/ zero-shot	w/ fine tuning
Validation Accuracy (r1)	32.8%	47.6%
Validation Accuracy (r2)	32.8%	47.6%
Validation Accuracy (rL)	32.8%	47.6%
Testing Accuracy (r1)	32.8%	48.6%
Testing Accuracy (r2)	32.8%	48.6%
Testing Accuracy (rL)	32.8%	48.6%

Data Preprocessing
For each row in the jsonl file, we construct training examples as below:

[EOS] + fact1 + question stem + [A] + choice A text + … + [D] + choice D Text + [EOS] + [correct choice][correct choice text] + [EOS]

An input example is constructed by concatenating "fact1" with the question stem and all the text from the choices as shown above. This is then wrapped by eos tokens, and we then append the choice text from the correct choice and then another eos token. The input examples are then tokenized, shuffled and passed into the model through a trainer.

Model
We use the pretrained "gpt2" model from HuggingFace as a base model. We then used the tokenizer from "gpt2" to resize the token embeddings. We also used a Trainer from Transformer for the main training and validation loop to perform fine tuning on the model. The final prediction is obtained by doing beam search over the generated output and finding the "[correct choice][correct choice text]" substring within the generated output.

Hyperparameters
For the tokenizer, we use parameters return tensors='pt', padding "max_length" with max_length=128. We use the Trainer module from Transformers with hyperparameters, per_device_train_batch_size=16, per_device_eval_batch_size=16, evaluation_strategy="steps", eval_steps = 2_00, logging_steps=2_00, gradient_accumulation_steps=2, num_train_epochs=8, weight_decay=0.1, warmup_steps=5_00, lr_scheduler_type="cosine", learning_rate=5e-4, _steps=5_00, fp16=True, push_to_hub=False.

Training loop
We used the Trainer from Transformer library, training on the entire tokenized training set and validation set. For the trainer parameters, we used the pretrained "gpt2" tokenizer, DataCollatorForLanguageModeling with mlm=False for the data_collator.

Evaluation
The model is evaluated using Rouge1, Rouge2, RougeL scores, which are computed using the "evaluate" library. Evaluation is performed on both validation and test set post training.

Zeroshot
To do zeroshot training and inference, we simply use the output from gpt2 without any finetuning, and again use beam search to find the generated output.