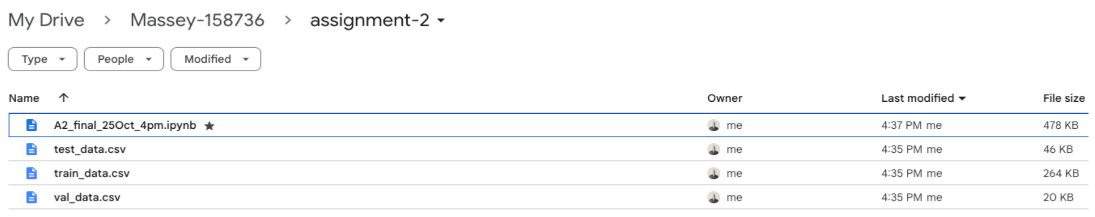
Massey University

158736 Advanced Machine Learning

Travel-domain text classification & fine-tuning using LLMs

Luis Vieira [23012096]

Google Colab Assignment-2 folder:


My Drive > Massey-158736 > assignment-2

Name	Owner	Last modified	File size
A2_final_25Oct_4pm.ipynb ★	me	4:37 PM me	478 KB
test_data.csv	me	4:35 PM me	46 KB
train_data.csv	me	4:35 PM me	264 KB
val_data.csv	me	4:35 PM me	20 KB

<u>Model Selection</u>

1. Justify model selection. You may refer to research papers, leader boards, etc.

When selecting the best language model for my task, I focused on Phi 3.5 mini instruct and Gemma 2 9B instruct due to their balance of performance, size, that are able to be ran on free platforms like Colab and Kaggle leveraging unsloth. I initially experimented with a range of models (all instruct versions) from 1~4B parameters up to 9 billion, but these two models stood out for their effectiveness in the QA text classification task. The decision was based on a few-shot evaluation and reference to leaderboards like the Open LLM Leaderboard and benchmarks/LLM comparisons how models perform for QA tasks (eg.: GPQA).
Given its size, the Phi 3.5 mini instruct performed exceptionally well. Despite being a tiny model, it handled tasks impressively, far better than other identical size models with its 128K token context length, making it ideal for handling tasks like this with very limited resources. This model's high ranking on leaderboards reflects its strong performance in reasoning and task comprehension, even against larger models (e.g.: Llama 3.1 8B it). Using unsloth's quantized version allowed me to run these models in a

resource-constrained environment like Colab/Kaggle without a considerable performance loss.
On the other hand, the Gemma 2 9B instruct model, with its larger parameter size, excels in more complex reasoning tasks. Although it required more computational power and time, unsloth's quantization made it feasible to run (mind some Out Of Memory errors).
Ultimately, I would choose Phi 3.5 mini for its superior balance between performance, computational efficiency, and its ability to handle longer-context and multi-languages in tasks, while Gemma 2 would be a better alternative in case of better GPU availability and some budget.

Sources:

Unsloth. (2024). Unsloth AI Documentation. https://docs.unsloth.ai

Hugging Face. (2024). Open LLM Leaderboard - Model Rankings and Comparisons*.* https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

PAIR Code. (2024). LLM Comparator - Compare Language Models*.* https://pair-code.github.io/llm-comparator

Microsoft Tech Community. (2024). *Discover the New Multi-Lingual, High-Quality Phi 3.5 SLMs.* https://techcommunity.microsoft.com/t5/ai-azure-ai-services-blog/discover-the-new-multi-lingual-high-quality-phi-3-5-slms/ba-p/4225280

2. Report the following information about the model - tokenizer used, vocabulary size, token count in the pre-training dataset, context length, post-training strategies, dataset used for SFT and three other design choices (e.g. activation function, positional embedding type)

Microsoft's Phi 3.5 mini instruct is a model designed with 3.8B parameters, using a dense decoder-only transformer architecture. It supports a context length up to 131,072 tokens, which is ideal for long-context tasks, such as large question answering. The model was pre-trained on a large 3.3 trillion tokens dataset. It uses the LlamaTokenizer with a vocabulary size of 32,064 tokens. While the technical documentation does not clearly detail the activation function or positional embedding type for Phi 3.5-mini, its close relative Phi-3-small employs GeGLU activation and Maximal Update Parametrization (muP) for effective hyperparameter tuning and enhanced stability during training. Additionally, it likely uses grouped query attention, optimised by a novel blocksparse attention, for faster training and inference. Post-training, Phi 3.5-mini was fine-tuned using supervised fine-tuning (SFT), on both human-generated and synthetic data, and further refined through Direct Preference Optimization (DPO) to better align its responses with human preferences.

On the other hand, Gemma 2 9B instruct offers a different set of strengths. With a parameter size of 9B, it handles a shorter context length of 8,192 tokens, relying on Rotary Position Embeddings (RoPE) for better sequential data management. Unlike Phi, Gemma 2 alternates between local sliding window attention and global attention across its layers, further improving its efficiency in processing longer sequences. The model uses a much larger vocabulary size of 256,128 tokens and was pre-trained on 6 trillion tokens, from a variety of structured and unstructured data sources, such as web documents and code. After pre-training, Gemma 2 was fine-tuned using SFT, alongside Reinforcement Learning from Human Feedback (RLHF), ensuring its responses meet human-like standards. Like Phi, it employs GeGLU activation, combined with grouped query attention, which replaces the traditional multi-head attention mechanism to improve computational efficiency.

Sources:

Microsoft. (2024). *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone.* https://arxiv.org/pdf/2404.14219

Microsoft. (2024). *Phi-3.5 Mini Instruct Model Card.* Retrieved from https://huggingface.co/microsoft/Phi-3.5-mini-instruct

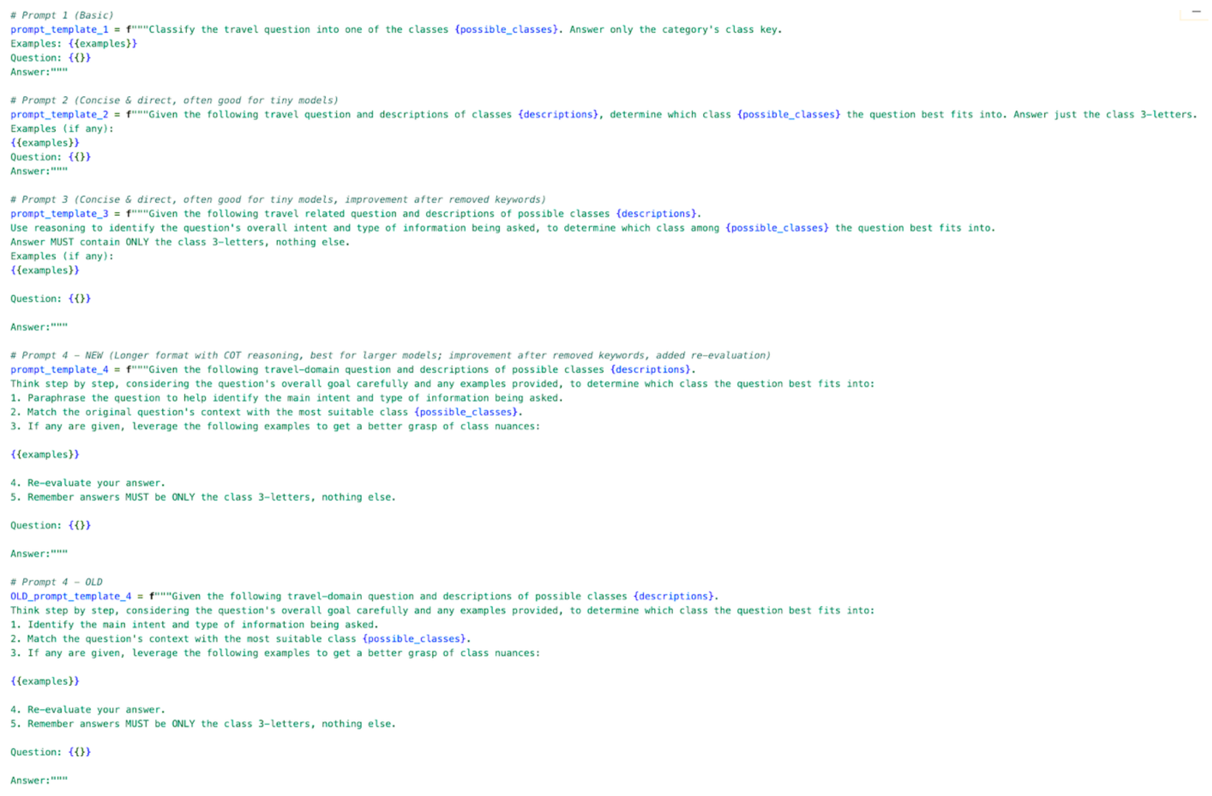
Gemma Team, Google DeepMind. (2024). *Gemma 2: Improving Open Language Models at a Practical Size.* arXiv. https://arxiv.org/abs/2408.00118

Prompt Engineering Guide. (2024). *Gemma 2 Overview.* https://www.promptingguide.ai/models

Google Developers Blog. (2024). *Gemma Explained: An Overview of the Gemma Model Family Architectures.* https://developers.googleblog.com/en/gemma-explained-overview-gemma-model-family-architectures

Prompt Development

3. Briefly explain your prompt design choices (e.g. what prompt pattern you used, what specific information had to be added to get the desired output from the LLM).



```
# Prompt 1 (Basic)
prompt_template_1 = f"""Classify the travel question into one of the classes {possible_classes}. Answer only the category's class key.
Examples: {{examples}}
Question: {{}}
Answer:"""

# Prompt 2 (Concise & direct, often good for tiny models)
prompt_template_2 = f"""Given the following travel question and descriptions of classes {descriptions}, determine which class {possible_classes} the question best fits into. Answer just the class 3-letters.
Examples (if any):
{{examples}}
Question: {{}}
Answer:"""

# Prompt 3 (Concise & direct, often good for tiny models, improvement after removed keywords)
prompt_template_3 = f"""Given the following travel related question and descriptions of possible classes {descriptions}.
Use reasoning to identify the question's overall intent and type of information being asked, to determine which class among {possible_classes} the question best fits into.
Answer MUST contain ONLY the class 3-letters, nothing else.
Examples (if any):
{{examples}}

Question: {{}}

Answer:"""

# Prompt 4 - NEW (Longer format with COT reasoning, best for larger models; improvement after removed keywords, added re-evaluation)
prompt_template_4 = f"""Given the following travel-domain question and descriptions of possible classes {descriptions}.
Think step by step, considering the question's overall goal carefully and any examples provided, to determine which class the question best fits into:
1. Paraphrase the question to help identify the main intent and type of information being asked.
2. Match the original question's context with the most suitable class {possible_classes}.
3. If any are given, leverage the following examples to get a better grasp of class nuances:

{{examples}}

4. Re-evaluate your answer.
5. Remember answers MUST be ONLY the class 3-letters, nothing else.

Question: {{}}

Answer:"""

# Prompt 4 - OLD
OLD_prompt_template_4 = f"""Given the following travel-domain question and descriptions of possible classes {descriptions}.
Think step by step, considering the question's overall goal carefully and any examples provided, to determine which class the question best fits into:
1. Identify the main intent and type of information being asked.
2. Match the question's context with the most suitable class {possible_classes}.
3. If any are given, leverage the following examples to get a better grasp of class nuances:

{{examples}}

4. Re-evaluate your answer.
5. Remember answers MUST be ONLY the class 3-letters, nothing else.

Question: {{}}

Answer:"""
```

When designing my prompts, I focused on balancing clarity and more complex reasoning while considering model size. Tests were initially conducted with temperature set to 0.1 on Kaggle's P100 GPU, using eight random sample questions. I tested a range of models using both Hugging Face (HF) [Gemma 2 2B & 9B, Llama 3.2 3B, Phi 3.5 mini, Qwen 2.5 3B] and some unsloth's 4bit model versions [same models except Llama 3.1 8B] to ensure results were identical.
Initially, Prompt 1 with minimal input, performed poorly for most models. I then moved to Prompt 2, which added concise class descriptions, leading to an improvement and was the best prompt with 42.86% accuracy for Gemma 2B, but still fell short for larger models.

A key improvement came with Prompt 3 and 4, where I refined the prompt no longer asking the model to focus on keywords and instead capturing the question's main intent. This helped Phi 3.5 mini instruct achieve 85.71% and later 90% accuracy, depending on the seed.
The biggest improvement was the introduced Chain of Thought (CoT) reasoning (mainly for prompt 4), a step-by-step approach that allowed models like Gemma 2 9B to reach 90-100% accuracy on the sample. This structured method, including a re-evaluation step, helped the model reason more effectively, especially for complex classification tasks.
These design choices reflect the need to balance prompt complexity with model capacity, a principle supported by research on CoT reasoning, which enhances model reasoning and performance, particularly in QA tasks.

Sources:

Prompting Guide. (2024). Prompting Techniques.
https://www.promptingguide.ai/techniques

Long, J. (2023). *Large Language Model Guided Tree-of-Thought*. arXiv.
https://arxiv.org/abs/2305.08291

Wei, J. et al. (2022). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.* arXiv. https://arxiv.org/abs/2201.11903

Shum, K. et al. (2023). *Automatic Prompt Augmentation and Selection with Chain-of-Thought from Labeled Data*. arXiv. https://arxiv.org/abs/2302.12822

Sahoo, P. *et al*. (2024). *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. arXiv. https://arxiv.org/abs/2402.07927

DotTxt AI. (2024). Structured text generation and robust prompting for language models.
https://dottxt-ai.github.io/outlines

Zero-shot and few-shot testing

4. What values did you select for temperature and top_p? Why did you choose those values?

Given that this is a classification task of travel-related questions, I experimented with a variety of settings, ultimately settling on temperature = 0.1 and top_p = 0.9 with top_k = 5. I chose these values to optimise

response accuracy and consistency while still allowing for a small room of diversity in the model's token selection process.
Initially, I tested the model with do_sample=False, commenting out temperature, top_p, and top_k, making the model fully deterministic. The results were identical to when I used temperature = 0.1, indicating that the model is already quite deterministic. My decision was based on the nature of classification tasks, where precision and consistency are essential, and there is limited need for creative variance in the outputs.
Given that there are only has 7 possible classes, I used top_k = 5 giving enough room for the model to select from the most relevant outputs without introducing too many unnecessary options. The max_new_tokens parameter was set very low to 5. This low token count ensures that the model's response is short, as it is expected to return only a 3-letter class code. Restricting the number of tokens prevents the model from generating irrelevant or extended answers, making it more likely to follow the prompt instructions. These configurations ensure that the model remains focused on classifying the travel questions, producing deterministic and accurate class outputs.

5. Observe the LLM outputs corresponding to the test set. What are your observations? Are there any recurring errors (e.g. hallucination)? Does few-shot prompting improve over zero-shot?

	Hugging Face			
Model	Gemma 2 2B it	Phi 3.5 mini it	Qwen 2.5 3B it	Gemma 2 9B it
Zero-shot	39.86	61.71	50.71	63.14
One-shot	48.14	61.29	58	61.43
Three-shot	52.71	62.86	60.14	67.14

	Unsloth (4bit)				
Model	Gemma 2 2B it	Phi 3.5 mini it	Qwen 2.5 3B it	Gemma 2 9B it	Llama 3.1 8B it
Zero-shot	42.86	62.71	45.29	69.71	49.43
One-shot	50.14	61.14	52.86	67.14	60.14
Three-shot	51.43	63.43	56.43	70.71	60.29

As mentioned previously, I tested a range of capable tiny and small LLMs available on HF and quantized versions from unsloth.
Overall, I noticed a clear improvement from zero-shot to three-shot evaluations across all models, with three-shot generally outperforming zero-shot. However, the one-shot results were inconsistent and didn't always improve upon zero-shot. I believe this might be due to the relevance and clarity of the sample question used in one-shot settings, like if the sample question is tricky or ambiguous, it could confuse the model and lead to misclassifications.
The misclassification analysis (Phi 3.5 mini) showed that models struggle with similar classes, such as confusing Food (FOD) with Entertainment (ENT), or Things to Do (TTD) with Weather (WTH). The high recall but low precision in Transport (TRS) indicates overprediction, while low recall in Travel Guide (TGU) suggests underprediction. Models tend to over rely on specific keywords without fully understanding the context.
For example:

“Which restaurants play bossa nova music in Rio?”

The model likely focused on "restaurants", associating it with FOD, and overlooked that the main intent is about music entertainment (ENT).

“What are the good bars in Caleta with entertainment?”

The model focused on "entertainment", predicting ENT, but the question is seeking recommendations for bars (places to eat and drink) so it should be classified under FOD.
I didn't observe any hallucinations where the model included a completely different class, even when a default “UNK” (Unknown) class was provided. Currently, the default is set to TGU as it is the largest represented class.
Note that unsloth models show better results because HF models were not tested on improved prompt iterations, using the same prompt results gave non-quantized models a 1~5% higher results. This suggests that refining prompts and providing clearer examples can enhance model accuracy.
To further improve performance, it's crucial to refine class definitions, clarify distinctions between overlapping categories, and help models understand the overall intent of questions rather than focusing solely on keywords, thus the reason why I removed the focus on keywords from the prompts.

Supervised Fine-tuning

6. What are the values you selected for the following hyper-parameters and why? Lora_r, use_4bit, fp16.

For the Fine-Tuning I used unsloth's quantized versions (use_4bit was set to True) of the models, which generally reduce memory usage while maintaining performance. Quantization allowed larger models like Gemma 2 9B and Llama 3.1 8B to run in resource-constrained environments, such as Colab's free-tier T4 and Kaggle's P100 or T4x2 GPUs, without overwhelming GPU's vRAM memory (barely). However, I observed a 1~5% performance drop when comparing these quantized models to the non-quantized versions available on Hugging Face, in pre-training (did not test in SFT).
In terms of precision, I set `dtype=None` and `fp16=not is_bfloat16_supported()`, to allow the system to auto-detect the most appropriate precision type. Since bfloat16 isn't supported on T4 and P100 GPUs, the unsloth pipeline defaults to fp16 for faster computations and reduced memory usage on these GPUs, which is important for handling large datasets and models in limited memory environments like the Colab's free tier (capped at 15Gb).
Lastly, I selected `lora_r=16` to balance memory efficiency with performance during fine-tuning. While higher LoRA ranks might increase fine-tuning performance slightly, they would also increase memory consumption, which is why 16 was a suitable choice, particularly for mid-sized models like Phi 3.5 mini and larger Gemma 2 9B that often maxed out available GPU (or led to Out Of Memory errors).
These hyperparameter choices ensured efficient use of resources, enabling me to run large models within free-tier environments while minimising performance loss.

7. Observe the LLM outputs to the test set. Compare your observations with those reported in Q4. Has the model improved its task performance with SFT?

After applying SFT (ORPO) fine-tuning, I observed a large improvement in the model's performance on the test set. The zero-shot accuracy increased from 59% to 82.86% for Phi 3.5 mini, and 69.71% to 87.71% for Gemma 2 9B, showing a substantial leap in its ability to correctly classify questions. Comparing the misclassifications before and after fine-tuning, I noticed that the model has become better at differentiating between similar classes, which was a major issue earlier. Previously, the model often confused overlapping classes and seemed to rely heavily on keywords without fully understanding the context.
After fine-tuning, while some misclassifications still occur, they are fewer and of a different nature. The model now shows a better understanding of the context and intent behind questions. However, I observed that it tends to overpredict the Travel Guide (TGU) class in certain cases:

"Would 700 pounds be enough to spend for a week in Sunny Beach?"

The model predicted TGU, whereas the correct class is TTD. It seems to associate budgeting questions with general travel guidance, even when the question is about planning activities.

"Is it best to go all-inclusive or just half board in Maldives?"

Here, the model predicted TGU instead of Accommodation (ACM). It might focus on the decision-making aspect, classifying it under travel guidance rather than recognising that the question is about accommodation options. Although the overprediction of TGU remains an issue, potentially related to being used as default class, the model has significantly improved its task performance overall. The misclassifications are less frequent, and it demonstrates a better grasp of context and intent. To further enhance performance, it's important to address class imbalance, refine class definitions, perhaps add synthetic data and difficult cases to help the model better distinguish between similar classes by focusing on the overall meaning of questions rather than specific keywords.

8. Report final results.

Model	**UNSLOTH** (4bit)		
	Gemma 2 2B it	Phi 3.5 mini it	Gemma 2 9B it
Zero-shot	42.86	62.71/59/59	69.71
One-shot	50.14	61.14/62.43/61.86	67.14
Three-shot	51.43	63.43/63/60.71	70.71
ORPO-prompt 3	--	80.29~**82.86**[1]	--
ORPO-prompt 4 old	54.29	81.57	**87.43 (87.71)**[2]
ORPO-prompt 4 new	--	**81.71**	87.14

Note. Pre-training: best prompt results (prompt 3/4old/4new), prompt 4 old for Gemma's.
Fine-Tuning uses a learning rate of 0.0002 and 100 steps, except #1 (lr 0.0001) and #2: (200 steps).

I evaluated three models (Gemma 2 2B it, Phi 3.5 mini it, and Gemma 2 9B it) using various prompting techniques and fine-tuning methods, including ORPO, which integrates SFT and Direct Preference Optimisation (DPO).
The results demonstrate a significant improvement in classification accuracy when using optimised prompts for training through ORPO, to fine-tune the models effectively. For instance, Gemma 2 9B achieved the highest accuracy of 87.43% using ORPO-prompt 4 old, compared to 69.71% in the zero-shot pre-training. Similarly, Phi 3.5 mini improved to 82.86% (changed learning rate from 0.0002 to 0.0001) with ORPO fine-tuning and prompt engineering, much better than other tiny LLMs and not too far behind Gemma 2 9B with less than half parameter size, and much faster train and output times.
These findings highlight the effectiveness of both prompt engineering and fine-tuning in enhancing model performance and show there is room for hyperparameter optimisation. By combining SFT with preference optimisation, ORPO allows models to better understand and classify questions based on improved prompts. The larger model, Gemma 2 9B it, consistently outperformed smaller ones, suggesting that increased model capacity contributes to higher accuracy.
Same as with learning rate, fine-tuning steps also influenced performance, with most models fine-tuned for 100 steps and one instance using 200

steps. However, the gains from additional fine-tuning appeared less significant than those achieved through prompt optimisation.
My goal was to compare the effectiveness of fine-tuning the best tiny (Phi 3.5 mini) and small size (Gemma 2 9B) models with limited resources and minimal tweaking of model hyperparameters to assess how much their classification performance could be improved under these constraints.
In summary, applying ORPO (SFT + DPO) leads to large improvements in classification accuracy, and tailored prompts and larger model sizes are key factors in achieving higher performance.

Additional Reading

9. Suppose you want to further optimise your model and you decide to use preference optimization. Discuss two preference optimisation techniques that are extensions of DPO. You need to highlight how those techniques overcome the limitations in the DPO model.

Two key extensions of DPO are ORPO (Odds Ratio Preference Optimisation) and KTO (Kahneman-Tversky Optimisation). These methods are designed to overcome the limitations of DPO, particularly in handling preference intensity, cognitive biases, and ensuring more stable training outcomes.
DPO, used for aligning LLMs with human preferences through pairwise comparisons, has several shortcomings. First, it is highly reliant on the initial fine-tuning of the model through supervised training, meaning that if the model isn't well aligned at the outset, DPO struggles to refine preferences effectively. Additionally, DPO tends to prioritise avoiding negative responses rather than maximising preferred ones, leading to imbalanced optimisation. It also assumes that all preferences are of equal strength, which is an oversimplification of real human judgments. Finally,

DPO's training process can become unstable, particularly when preferences are closely matched.
In my fine-tuning pipeline, I used ORPO, a method introduced by Hong *et al.* (2024), which improves DPO by using odds ratios rather than simple probabilities to represent preference strengths. This approach gives more weight to stronger preferences, making the optimisation process more robust and the model more effective in distinguishing between varying preference intensities. ORPO also addresses the instability issue by ensuring a more stable training process.

KTO, as proposed by Ethayarajh et al. (2024), extends DPO by incorporating Prospect Theory to better account for human cognitive biases, such as loss aversion and probability weighting. Unlike DPO, which assumes rational decision-making, KTO models how individuals evaluate outcomes relative to reference points. This makes KTO better suited to real-world decision-making, where human behaviour is often influenced by psychological factors.

In conclusion, ORPO and KTO significantly enhance DPO by addressing its limitations, particularly in preference strength calibration and incorporating cognitive biases. These methods provide a more realistic and stable approach to preference optimisation, as demonstrated in my fine-tuning pipeline.

Sources:

Rafailov, R., *et al.* (2024). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv. https://arxiv.org/pdf/2305.18290

Feng, D., *et al.* (2024). *Towards Analyzing and Understanding the Limitations of DPO: A Theoretical Perspective*. arXiv. https://arxiv.org/pdf/2404.04626

Hong, j., *et al.* (2024). *ORPO: Monolithic Preference Optimization without Reference Model*. arXiv. https://arxiv.org/abs/2403.07691

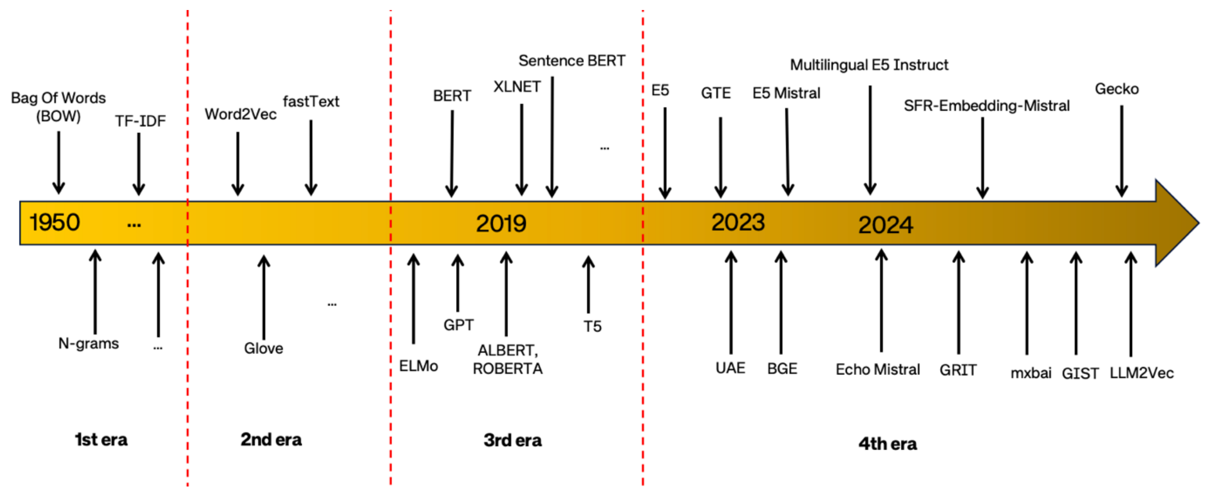
Ethayarajh, K., *et al.* (2024). *KTO: Model Alignment as Prospect Theoretic Optimization*. arXiv. https://arxiv.org/abs/2402.01306

10. Suppose you are to incorporate a RAG system with the LLM for it to support more recent sources. Three important selection decisions you need to make are: 1. The vector database, 2. Sentence embedding technique and 3. Retrieval technique. For each of these three components, identify two possible options and briefly discuss their features.

When incorporating a Retrieval-Augmented Generation (RAG) system, we need to make crucial decisions regarding the vector database of choice, the embedding and retrieval approaches.

For the database, two of the leading options are Milvus and Elasticsearch. Milvus is a scalable, open-source vector database that excels at managing large-scale unstructured data, thanks to its distributed architecture supporting multiple index types (IVF_FLAT, IVF_SQ8). Also supports real-time hybrid retrieval (dense and sparse) with high concurrency, ideal for complex RAG workflows. Whereas Elasticsearch leverages its inverted index structure alongside dense vector capabilities through the KNN search plugin, allowing for composite queries that combine traditional text matching with vector similarity - essential for complex travel queries involving both specific terms and semantic meaning.

When selecting a sentence embedding technique for a RAG system in the travel domain, several top-performing models offer distinct advantages. Nvidia's NV-Embed-v2, currently leading the MTEB (Massive Text Embedding Benchmark), leverages latent-attention pooling combined with a decoder-only LLM architecture, optimised through a two-stage instruction tuning method. For faster and smaller-scale applications, Stella_en_400M_v5 provides competitive performance with a reduced computational footprint. On the other hand, SBERT (Sentence-BERT) utilises a siamese and triplet network architecture to derive semantically meaningful sentence embeddings. It remains a robust choice, with models like 'all-mpnet-base-v2' using efficient mean pooling to deliver high-fidelity sentence representations. E5, developed by Microsoft, employs contrastive learning with the InfoNCE loss function, making it optimised for retrieval tasks, particularly useful when aligning travel-related questions with detailed destination descriptions. More recently, we find new methodologies like “Starbucks”, which enhances 2D Matryoshka Embeddings through curriculum learning with controlled negative sampling and adaptive margins, enabling more stable training while preserving dynamic dimensionality scaling. (Zhuang, *et al.*, 2024)



Source: (Cao, 2024)

The primary retrieval methods often include Hybrid search (dense and sparse), ANN search, and Graph-Based retrieval. Hybrid Search integrates dense vector retrieval with BM25 keyword-based search, combining results from both methods. This is likely ideal for travel-related QA requiring precise keyword matches and semantic context. ANN Search, especially with HNSW (Hierarchical Navigable Small World) graphs, enables efficient similarity searches in high-dimensional spaces, reducing latency while maintaining high recall. Graph-Based Retrieval uses knowledge graphs to link travel entities such as destinations, hotels, and activities, offering deeper contextual connections. Algorithms like Personalized PageRank and random walk with restart are used to explore these relationships and deliver interconnected travel information.

Sources:

Superlinked. (2024). Vector Database Comparison.
https://superlinked.com/vector-db-comparison

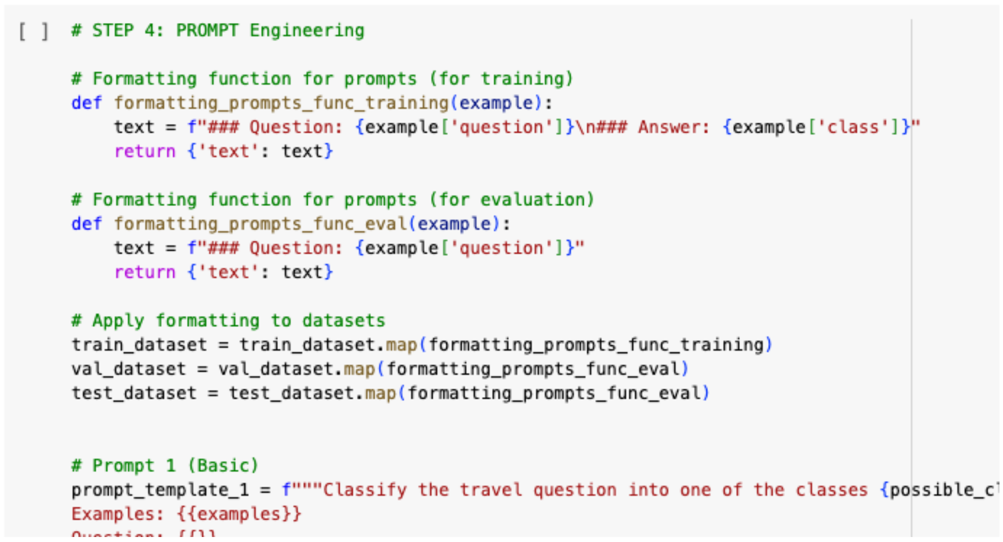
Zhuang, S., *et al*. (2024). *Starbucks: Improved Training for 2D Matryoshka Embeddings*. arXiv. https://arxiv.org/pdf/2410.13230

Cao, H. (2024). *Recent advances in universal text embeddings: A Comprehensive Review of Top-Performing Methods on the MTEB Benchmark*. arXiv. https://arxiv.org/pdf/2406.01607

Snapshots of the code used to convert the dataset to instruction format

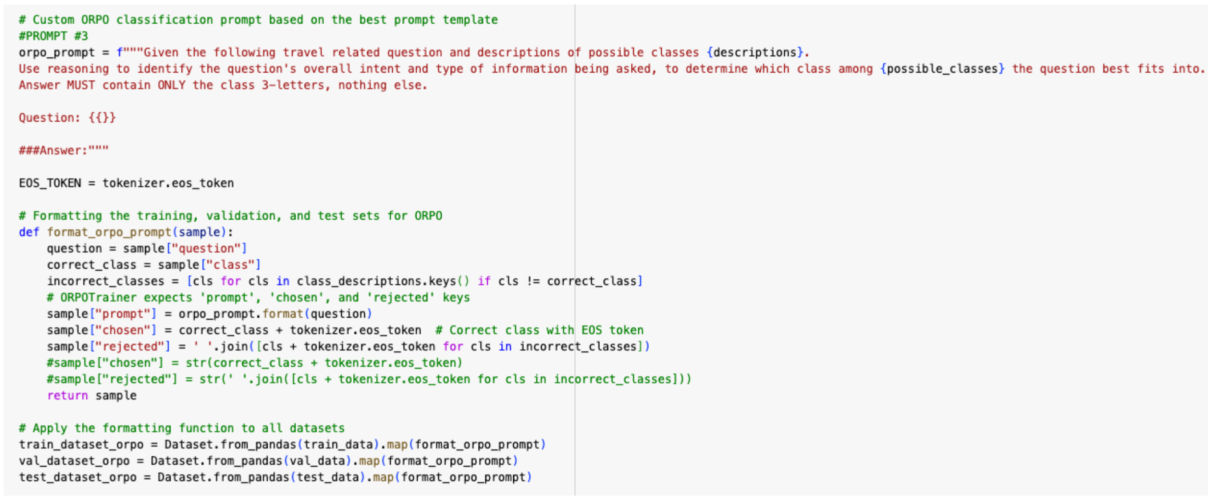
Pre-Training:

Prompt Engineering & Evaluation



```
[ ]  # STEP 4: PROMPT Engineering

     # Formatting function for prompts (for training)
     def formatting_prompts_func_training(example):
         text = f"### Question: {example['question']}\n### Answer: {example['class']}"
         return {'text': text}

     # Formatting function for prompts (for evaluation)
     def formatting_prompts_func_eval(example):
         text = f"### Question: {example['question']}"
         return {'text': text}

     # Apply formatting to datasets
     train_dataset = train_dataset.map(formatting_prompts_func_training)
     val_dataset = val_dataset.map(formatting_prompts_func_eval)
     test_dataset = test_dataset.map(formatting_prompts_func_eval)


     # Prompt 1 (Basic)
     prompt_template_1 = f"""Classify the travel question into one of the classes {possible_c
     Examples: {{examples}}
```

SFT (ORPO):



```
# Custom ORPO classification prompt based on the best prompt template
#PROMPT #3
orpo_prompt = f"""Given the following travel related question and descriptions of possible classes {descriptions}.
Use reasoning to identify the question's overall intent and type of information being asked, to determine which class among {possible_classes} the question best fits into.
Answer MUST contain ONLY the class 3-letters, nothing else.

Question: {{}}

###Answer:"""

EOS_TOKEN = tokenizer.eos_token

# Formatting the training, validation, and test sets for ORPO
def format_orpo_prompt(sample):
    question = sample["question"]
    correct_class = sample["class"]
    incorrect_classes = [cls for cls in class_descriptions.keys() if cls != correct_class]
    # ORPOTrainer expects 'prompt', 'chosen', and 'rejected' keys
    sample["prompt"] = orpo_prompt.format(question)
    sample["chosen"] = correct_class + tokenizer.eos_token  # Correct class with EOS token
    sample["rejected"] = ' '.join([cls + tokenizer.eos_token for cls in incorrect_classes])
    #sample["chosen"] = str(correct_class + tokenizer.eos_token)
    #sample["rejected"] = str(' '.join([cls + tokenizer.eos_token for cls in incorrect_classes]))
    return sample

# Apply the formatting function to all datasets
train_dataset_orpo = Dataset.from_pandas(train_data).map(format_orpo_prompt)
val_dataset_orpo = Dataset.from_pandas(val_data).map(format_orpo_prompt)
test_dataset_orpo = Dataset.from_pandas(test_data).map(format_orpo_prompt)
```