Task 1 - GNN

Conclusion

- **Stop Training at Peak Performance**
    - The model peaked at around 150 epochs, after which it collapsed.
    - Solution: Use early stopping. If validation AUROC stops improving for 10-20 epochs, stop training.
- **Reduce Overfitting**
    - The model likely memorized training data instead of learning useful patterns.
    - Fixes:
        - Add dropout (0.3-0.5) in GAT layers.
        - Apply L2 regularization (weight decay around 1e-4).
        - Reduce the model size or attention heads if it's too complex.
- **Address Class Imbalance**
    - Recall = 1.0 but Precision = 0.76 indicates the model is predicting mostly one class.
    - Fixes:
        - Use a weighted loss function (class_weight in CrossEntropyLoss).
        - Apply oversampling/undersampling to balance BBB+ and BBB-.
        - Try focal loss instead of standard cross-entropy.
- **Adjust Learning Rate & Training Strategy**
    - The loss function flattens out, meaning the model isn't improving anymore.
    - Fixes:
        - Reduce learning rate dynamically. Use a scheduler (ReduceLROnPlateau).
        - Start with an lr = 0.001, then lower to 0.0003 when validation AUROC plateaus.
        - Train for fewer epochs (around 150).
- **Summary**
- The best AUROC (around 0.92) was real but not stable.
- The final AUROC = 0.51 means the model collapsed after learning useful patterns.
- Training should be stopped earlier, regularization added, and class imbalance addressed.

1. Describe how different message-passing strategies (e.g., mean aggregation, sum aggregation, attention-based) affect model performance.

Graph Neural Networks (GNNs) use message-passing to allow nodes (atoms, in our case) to learn from their neighbours (other connected atoms). The way these messages are aggregated significantly impacts model performance, and different strategies have different advantages and limitations.

Mean aggregation takes the average of neighbouring node features and is useful when graphs have a balanced structure with nodes having similar connectivity. However, in molecular graphs where some atoms have many more connections than others, mean aggregation tends to dilute important features. This can lead to a model that fails to distinguish structurally different molecules because all atoms start looking too similar. In our model, which uses Graph Attention Networks (GAT), mean aggregation would have likely failed because molecules have varied bond connectivity, meaning that treating all neighbours equally would not capture their actual importance in determining permeability.

Sum aggregation, on the other hand, totals the features from all neighbouring nodes. This approach ensures that larger molecules with more neighbours retain strong signals, but it also introduces risks when graphs have highly connected nodes. If a single node receives too much information from too many neighbours, its representation can become overloaded and lose meaningful distinction. This is particularly relevant in molecules where central atoms connect to multiple functional groups, potentially making them dominant in feature propagation. If our model had used sum aggregation, it might have struggled with very large molecules, where overly connected nodes could overpower useful features.

Attention-based aggregation, as used in GAT, solves these problems by dynamically learning the importance of each neighbour. Instead of averaging or summing blindly, attention assigns higher weights to significant connections. In molecular graphs, this means that atoms in functional groups related to BBB permeability receive more importance than atoms in irrelevant parts of the molecule. Our model initially performed well with this approach, achieving an AUROC of around 0.82, which indicates that it was correctly identifying critical atomic interactions. However, the AUROC later collapsed to 0.51, suggesting that the attention mechanism overfitted to patterns in the training data and failed to generalize. This could have been caused by excessive focus on a small subset of features, ignoring broader molecular structures. One possible fix is to reduce the number of attention heads to prevent over-complexity or introduce dropout in the attention layers to regularize learning.

2. What is the role of edge features in GNNs? How would you modify a basic GNN to incorporate them?

Edges in a molecular graph represent bonds between atoms, and in our current model, these are treated equally without distinguishing between single, double, triple, or aromatic bonds. However, in chemistry, the type of bond significantly impacts molecular properties. A triple bond between two carbon atoms contributes very different electronic and structural properties compared to a single bond between carbon and oxygen. Our model, by ignoring these bond features, might have failed to distinguish between molecules that are

structurally similar but chemically different in terms of how they interact with the blood-brain barrier.

To incorporate edge features, we need to modify the data representation so that bonds have their own attributes. In PyTorch Geometric, this means adding an edge_attr tensor containing bond properties such as bond type, bond length, or aromaticity. The current implementation of our GAT model does not use this, meaning every bond is treated equally regardless of its actual effect on permeability. By modifying the GATConv layers to accept edge_dim, we can allow the model to learn how different bond types influence permeability predictions. Additionally, replacing GATConv with GatedGraphConv could enable explicit modelling of edge interactions, allowing information to propagate more meaningfully across a molecule.

The expected improvement from incorporating edge features is twofold. First, it would improve chemical accuracy, ensuring that the model recognizes why certain molecules pass the BBB while others do not. Second, it would help in generalization, as the model would no longer rely solely on atomic connections but also on the nature of those connections. Given that our model collapsed in later training epochs, this suggests that it overfitted to atomic properties alone and adding bond information could provide additional structural context to make it more robust.

3. Given a highly connected molecular graph, what challenges arise in GNN training, and how would you address them?

Molecular graphs can be highly connected, meaning that some atoms are linked to many others, forming dense substructures. While message-passing in GNNs allows information to propagate across the graph, excessive connectivity can lead to over-smoothing, where all nodes start to look alike. In our training, we likely encountered this problem as AUROC initially increased but later flattened and collapsed. This suggests that after multiple layers of message-passing, nodes lost their distinctiveness, making it harder for the model to differentiate molecules. One way to prevent this is by reducing the number of GAT layers, as fewer propagation steps prevent excessive feature mixing.

Another challenge is the vanishing of meaningful information in large graphs. In molecular graphs, critical information often resides in localized substructures, such as rings or functional groups. However, if messages pass too many times, important signals can get lost as information diffuses across the entire graph. Our training instability suggests that this might have been an issue, as the model initially learned well but later lost performance after too many epochs. A good solution is to introduce skip connections, allowing important information to bypass certain layers and reach deeper ones without degradation. This is particularly useful in large molecules where distant atoms might need to retain independent features rather than blending with the entire structure.

Another major problem in training GNNs on molecular data is computational cost. Unlike standard deep learning models that work with fixed-sized inputs, GNNs must process graphs of varying sizes. Large graphs require significantly more computations per node, especially when using attention-based models like GAT that compute pairwise attention scores. This

can lead to slow training and high memory consumption, which can make the model difficult to scale. In our case, training performance became unstable after 150-180 epochs, which could be a symptom of inefficient graph computations. A practical solution is to use GraphSAGE instead of full GAT, which samples a subset of neighbors instead of computing attention across all nodes. This reduces training overhead while maintaining performance.

Class imbalance is another issue that likely impacted our training. The final model showed perfect recall (1.0) but poor precision (0.76), suggesting that it was predicting one class much more often than the other. If most molecules in the dataset are BBB+ (permeable), the model may have learned to simply classify everything as permeable, achieving high accuracy but low AUROC. This aligns with the fact that AUROC dropped to 0.51, meaning it lost any real predictive ability. A potential fix is to balance the dataset using resampling techniques or introduce class-weighted loss functions to prevent the model from biasing towards one dominant class.

Overall, the challenges in training GNNs for molecular data—over-smoothing, vanishing information, computational inefficiency, and class imbalance—were all reflected in our results. The key takeaways are that we need to limit message-passing to prevent excessive feature blending, introduce edge features to capture bond information, and address class imbalances to ensure that the model learns a balanced decision boundary. With these changes, our model should generalize better and avoid the performance collapse that occurred during training.

Task 2 – Trasnformers

Conclusion

The assignment conducted demonstrates the critical role that multi-head self-attention and positional encodings play in BERT's ability to model language effectively. By systematically analysing how disabling attention heads and removing positional encodings impact entropy, attention distance, and hidden state differences, we uncover the mechanisms that allow BERT to structure language representations.

Key Observations from Assignment Results

1. **Original Model Performance**:
    - Attention entropy across sentences ranged between 1.91 and 2.46, indicating a well-balanced focus.
    - Average attention distance was relatively low, ranging from 4.65 to 11.46, meaning that words primarily attended to nearby tokens while maintaining some long-range connections.
    - The heatmaps showed clear patterns where words attended to syntactically and semantically related words, confirming that BERT effectively learns linguistic dependencies.
2. **Disabling Attention Heads**:
    - Attention entropy increased in all cases, confirming that removing specialized attention heads results in more diffused attention.
    - Average attention distance increased slightly, indicating that some long-range dependencies became more prominent as local connections weakened.
    - Heatmap analysis showed that some relationships were still preserved, but the attention became more scattered, meaning that BERT lost some ability to focus on key words.
    - Hidden state differences remained relatively small, suggesting that removing a few heads does not drastically change word representations but weakens the structured learning process.
3. **Removing Positional Encodings**:
    - Entropy increased more significantly, rising from an average of 2.31 to above 3.0, indicating a complete loss of structured attention.
    - Attention distance increased substantially from values as low as 4.65 up to 13.71, proving that words attended to distant and often unrelated tokens rather than maintaining meaningful sentence structures.
    - Hidden state differences were significantly larger than in the disabled heads assignment, confirming that removing positional encodings drastically alters the model's understanding of input text.
    - The heatmaps exhibited completely disorganized patterns, proving that BERT loses its ability to track word order without positional encodings.
4. **Combining Both Modifications**:
    - This led to the most severe degradation in model performance.
    - Entropy increased further to 3.10 and above, and attention distance reached its highest values (above 13), confirming complete disorder in token relationships.

- The hidden state differences were the largest among all assignments, proving that removing both attention specialization and positional awareness disrupts BERT's entire ability to structure language representations.
- The heatmaps showed almost randomized distributions of attention, meaning that the model had lost nearly all structured understanding of sentence context.

Final Interpretation and Conclusion

These findings confirm that multi-head self-attention and positional encodings are essential components of BERT's architecture. While disabling some attention heads weakens the model's contextual understanding, the removal of positional encodings completely breaks sentence structure comprehension. When both modifications are applied together, BERT loses its ability to process text meaningfully.

The structured nature of self-attention allows BERT to specialize different heads for different aspects of language learning, while positional encodings ensure that words retain a logical order. These two mechanisms work together to create strong language representations and removing them results in significant performance degradation.

1. Explain how multi-head self-attention improves Transformer performance. Why not just use a single attention head?

Multi-Head Self-Attention and Its Benefits
Multi-head self-attention (MHSA) plays a crucial role in improving the performance of Transformers by allowing the model to process multiple relationships between words in parallel. Unlike traditional single-head attention, which forces the model to focus on only one type of dependency at a time, multi-head attention enables different heads to capture different aspects of the input sequence. For instance, some heads might focus on grammatical relationships, while others might focus on semantic meaning. This ability to extract diverse patterns strengthens the model's understanding of context, leading to better predictions and generalization.

If a Transformer were to use only a single attention head, it would struggle to capture the full complexity of language. A single head would be forced to prioritize only one type of relationship, potentially ignoring subtle but important dependencies in the sentence. This limitation would make the model less flexible, as it would not be able to adjust its focus dynamically depending on the input. Additionally, relying on a single head increases the risk of overfitting, as the model might become overly reliant on a specific attention pattern rather than learning robust, generalizable representations.

Assignments with disabling some attention heads show that attention becomes more diffuse, meaning the model distributes its focus across more words rather than homing in on key dependencies. This suggests that multiple attention heads contribute to making attention more structured and meaningful. Without them, the model has difficulty concentrating on the most relevant words, which reinforces why multi-head self-attention is essential for maintaining clarity in information processing. The multiple heads work together to refine attention patterns, making the model more effective in capturing intricate word relationships.

Using only one attention head would force all linguistic roles into a single computation, making it harder for BERT to separate short-term dependencies from long-term ones. Our heatmaps showed that removing heads led to more scattered attention, meaning that different roles became merged into a less interpretable form.
Multi-head attention prevents over-compression of information, ensuring that BERT captures a diverse set of relationships in text. Without it, the model fails to distinguish complex patterns, which is why removing heads reduced its ability to structure meaning correctly.

2. If positional encodings were removed from a Transformer, what impact would this have on performance?

Transformers do not inherently understand the order of words in a sequence, unlike models such as RNNs that process data sequentially. Instead, they rely on positional encodings to introduce word order information. These encodings allow the model to differentiate between words that appear at different positions in a sentence, ensuring that the meaning of a phrase is correctly preserved. Without positional encodings, a Transformer would treat

all words as equally related, regardless of their order. This would be particularly problematic for tasks that depend on syntax, such as machine translation and text summarization, where word arrangement significantly affects meaning.

When positional encodings are removed, attention tends to spread out more broadly, making it harder for the model to establish local dependencies. Instead of focusing on nearby words that provide contextual clues, the model may end up attending to unrelated words at distant positions. This disrupts the natural structure of language, as words that should be closely connected may no longer be associated. Longer sentences are particularly vulnerable to this issue because they contain more dependencies across different words, and without positional encodings, the model loses its ability to determine which relationships are most relevant.

Assignment results confirm this expectation by showing that attention shifts towards more distant words when positional encodings are removed. This indicates that the model struggles to recognize natural language structure without positional guidance. The increase in entropy observed in these assignments further suggests that attention becomes more scattered, making it harder for the model to focus on critical elements of a sentence. This highlights the importance of positional encodings in ensuring that Transformers not only recognize word relationships but also understand the sequence in which words appear.

Observations from assignment:

- Entropy increased significantly, meaning that attention distributions became more chaotic.
- Average attention distance rose from around 4-7 to over 13, proving that the model started attending to distant, unrelated words instead of preserving sentence structure.
- Heatmaps showed complete disorder, confirming that without positional encodings, BERT treats sentences as unordered bags of words.

Expected Performance Impact
For Sequence-Dependent Tasks:

- **Language Translation**: Performance would likely deteriorate significantly as word order is critical for meaning.
- **Text Generation**: The model would struggle to produce coherent, grammatical text.
- **Syntactic Analysis**: Tasks requiring understanding of grammatical structure would suffer.

For Classification Tasks:

- **Sentiment Analysis**: May see less dramatic degradation if sentiment depends more on word presence than order.
- **Topic Classification**: Might maintain reasonable performance if topics can be identified from key terms regardless of order.

Unexpected Behaviours
Interestingly, removing positional encodings doesn't always cause catastrophic failure:

- Some Transformer models can implicitly learn positional information through training patterns.
- For certain tasks, the self-attention mechanism might develop a form of positional awareness through learned attention patterns.
- In some contexts, models without positional encoding generalize better to sequence lengths not seen during training.

Technical Consequences
Without positional encodings:
- All tokens at different positions with the same value would be indistinguishable to the model.
- Self-attention would focus purely on content-based relationships.
- The model would lose the ability to directly model distance between tokens.
- Attention patterns would be symmetric (token A attending to token B would be the same as B attending to A).

3. How Changing the Number of Attention Heads Affects Computational Cost and Model Performance

Attention heads are a crucial component of Transformer-based models like BERT and GPT, allowing them to process multiple aspects of input data in parallel. Each head operates independently, capturing different linguistic relationships such as syntax, semantics, and named entities. The number of attention heads directly affects both computational efficiency and model performance. While more heads improve contextual understanding, they also introduce greater computational costs. Conversely, fewer heads enhance efficiency but may weaken the model's ability to extract complex relationships.

Computational Cost of Attention Heads
Attention heads impact three key areas:
- Memory Usage: Each head maintains a separate set of attention weights, increasing overall memory requirements. The attention score matrix scales with batch size and sequence length, making memory consumption grow linearly with the number of heads. This can be a bottleneck for GPUs and TPUs with limited memory.
- Computational Complexity: More heads mean more floating-point operations (FLOPs) due to additional matrix multiplications for query, key, and value transformations. This leads to longer training and inference times, increasing computational demand.
- Parameter Efficiency: Despite higher computational costs, the total parameter count remains nearly constant because the dimensionality per head is adjusted accordingly. However, extremely low head counts can disrupt training stability, as fewer heads must handle more linguistic functions.

Effect of Attention Heads on Model Performance
- More Heads (8-12+):
    - Improves generalization by enabling different heads to focus on distinct linguistic patterns.

    - Enhances contextual representation, making models more effective for unseen data.
    - Diminishing returns beyond 8-12 heads, as additional heads provide little performance gain but increase computational cost.
- Fewer Heads (<8):
    - Speeds up inference and reduces memory usage, beneficial for low-resource environments.
    - Leads to weaker contextual encoding, as fewer heads must share more linguistic roles.
    - Attention becomes more diffused, reducing the model's ability to maintain structured word relationships.

Redundancy and Pruning of Attention Heads

Studies show that not all attention heads contribute equally to performance. Pruning unnecessary heads can optimize efficiency without significantly reducing accuracy:

- Lower layers (1-4): Up to 50% of heads can be pruned with minimal performance loss.
- Middle layers (5-8): 20-30% of heads can be removed without major degradation.
- Final layers (9-12): Only 10-15% of heads can be pruned before performance starts deteriorating.

These findings suggest that some heads are task-specific, and selectively removing redundant heads can reduce computational costs while maintaining accuracy.

Trade-offs and Practical Considerations

- More heads enhance the ability to learn complex dependencies but demand higher memory and processing power.
- Fewer heads improve efficiency and speed but may lead to oversimplified linguistic relationships.
- The ideal number of heads depends on hardware availability, dataset complexity, and specific NLP tasks.
- Assignments show that disabling too many attention heads causes the model to struggle with coherence, weakening performance.
- Positional encodings and attention heads work together removing both leads to less meaningful representations and disrupts natural language flow.

Conclusion

The number of attention heads is a critical hyperparameter in Transformer models, directly impacting efficiency, accuracy, and computational cost. Research suggests that 8-12 heads provide the best balance between performance and efficiency, as additional heads beyond this range offer minimal benefits. Pruning redundant heads can optimize model size and reduce resource consumption without significantly affecting accuracy. Ultimately, selecting the optimal number of heads depends on task complexity, computational resources, and model objectives.

Task 3 – RAG

Observations:

1. **Retrieval Method Comparison (BM25 vs Dense)**
    - BM25 tends to retrieve longer and more interpretable answers, while Dense retrieval occasionally provides shorter or vague responses.
    - In some cases, BM25 answers are consistent across different top_k values, whereas Dense retrieval results vary significantly when increasing top_k.

2. **Answer Quality and Relevance**
    - Some retrieved answers, such as *"artificial intelligence"* or *"Biomarker"*, are overly generic and lack specific insights.
    - The confidence scores for many responses are low (<0.5), indicating uncertainty in retrieval accuracy.
    - Responses related to Generative AI for image classification provide reasonable detail, but some retrieved answers (e.g., *"patch-clamp recordings and molecular dynamic simulations"*) seem out of context.

3. **Token Length and Answer Specificity**
    - Shorter answers (1-2 tokens) often lack specificity and may not provide useful information (e.g., *"AI"*, *"Biomarker"*).
    - Longer answers (5+ tokens) tend to be more meaningful and informative, but can still lack precision (e.g., *"innovative tools and technologies, including artificial intelligence"*).

4. **Class Imbalance in Confidence Scores**
    - There is a significant variance in confidence scores, even for similar questions and retrieval methods.
    - Some responses have high confidence but poor relevance, while others with lower confidence scores appear more accurate.

5. **Top-K Effect**
    - Increasing top_k does not always improve answer quality. In some cases, the retrieved answers remain the same (BM25), while for Dense retrieval, there are noticeable shifts in answer selection.

Conclusion & Recommendations:

1. **BM25 is More Stable**
    - BM25 provides more stable responses across different top_k values, making it a reliable choice for structured retrieval tasks.
    - However, it sometimes retrieves generalized answers that lack depth.

2. **Dense Retrieval Needs Refinement**
    - Dense retrieval produces more varied answers, but some results appear unrelated or out of context.
    - Using hybrid retrieval (BM25 + Dense) could balance stability and contextual richness.

3. **Improve Answer Specificity**
    - Many responses are too short or too vague, reducing their usefulness.
    - Implementing answer re-ranking based on contextual similarity and fine-tuning embeddings could enhance response relevance.

4. **Confidence Scores Do Not Always Indicate Accuracy**
    - Some low-confidence responses are actually better answers, while some high-confidence ones are irrelevant.
    - A manual review or an additional validation step could help filter out incorrect but high-confidence responses.

5. **Adaptive Top-K Strategies Needed**
    - Increasing top_k does not consistently improve retrieval performance.
    - A dynamic top_k selection based on question complexity could optimize the retrieval process.

Final Takeaway:
The retrieval pipeline needs hybrid optimization (BM25 + Dense), better re-ranking, and improved context filtering to ensure specific, relevant, and high-confidence answers.

1.How does retrieval quality impact the final generation in RAG-based models?

Retrieval-Augmented Generation (RAG) relies on retrieving relevant documents before generating responses. The quality of the retrieved content directly affects the accuracy, confidence, and completeness of the final output. If retrieval fails to find the right information, even the best language model cannot generate meaningful responses.

Retrieval Method Differences
RAG typically uses two retrieval approaches:

- **BM25 (Keyword-Based Retrieval):** Retrieves documents based on word matching. It tends to return **consistent results**, even when increasing the number of retrieved documents (**top_k**).
- **Dense Retrieval (Embedding-Based):** Uses **vector similarity** to find semantically relevant documents. It is **more sensitive** to the number of retrieved documents and often **varies significantly** between **top_k=5 and top_k=10**.

In my assignment, BM25 consistently retrieved similar answers, while dense retrieval produced different responses when more documents were retrieved. This indicates BM25 is reliable for factual consistency, whereas dense retrieval provides a broader range of perspectives.

Impact on Accuracy & Confidence
Better retrieval leads to more confident and accurate answers. This was evident in the metabolics study question:

- BM25 (top_k=10) → Confidence score: 64.6%
- Dense retrieval (top_k=5) → Confidence score: 0.8%

The confidence score dropped drastically when using dense retrieval with fewer documents. This means that if retrieval quality is poor, the final answer will be uncertain or incorrect.
Effect on Answer Completeness
The amount and relevance of retrieved information determine how detailed the generated response will be. If the retrieval system fails to find all necessary information, the response will be incomplete or vague.

This was visible in my results, where token length varied significantly based on retrieval quality.

- Some responses were 1-2 words long, indicating insufficient retrieved context.
- Others were 7-9 words long, providing more complete answers.

Consistency vs. Diversity in Responses
BM25's keyword-based approach retrieves the same documents even when the number of retrieved results increases. This makes it a good choice for factual consistency.
In contrast, dense retrieval shows more variation, meaning it introduces more diverse information. While this can be beneficial, it also leads to inconsistencies when asking the same question multiple times.

Impact of Retrieved Document Quantity

Increasing top_k (number of retrieved documents) had different effects depending on the retrieval method:

- **BM25:** Retrieved **the same answer**, whether **top_k was 5 or 10**.
- **Dense Retrieval:** Retrieved **completely different answers** when top_k increased (e.g., "predictive models" → "Biomarker").

This suggests BM25 is more stable, while dense retrieval requires careful tuning.

Practical Takeaways for RAG Optimization

1. Choose Retrieval Methods Based on Needs
    - For fact-based consistency: BM25 is more reliable.
    - For exploring diverse perspectives: Dense retrieval provides richer context.
2. Tune Retrieval Parameters
    - top_k should be adjusted carefully to avoid unstable answers.
    - A hybrid approach (BM25 + Dense retrieval) can combine stability with contextual depth.
3. Confidence Scores Indicate Retrieval Quality
    - Low confidence often signals poor retrieval, requiring retriever improvements, not just generator tuning.
4. For Healthcare Applications:
    - BM25 is better for reliable responses (e.g., diagnoses, treatment plans).
    - Dense retrieval is useful for exploratory research but requires careful validation.

Conclusion

The retrieval stage is the foundation of any RAG system. A weak retriever limits answer accuracy, no matter how powerful the generator is. My assignment shows that optimizing retrieval quality improves confidence, completeness, and reliability, making it a crucial area of focus when designing RAG systems.

2. Given a low-resource dataset, how would you fine-tune a retriever in a RAG pipeline?

I would approach fine-tuning a retriever for a low-resource dataset as follows:

Leveraging Pre-trained Models

My RAG implementation demonstrates using a pre-trained sentence transformer model ("all-MiniLM-L6-v2") for embedding. With limited data, I would:

- Start with a domain-specific pre-trained model (like BioBERT for healthcare data)
- Freeze most layers and only fine-tune the top layers to prevent overfitting
- Use the existing embedding architecture: embedding_model = SentenceTransformer("domain-specific-model")

Implementing a Hybrid Retrieval Strategy

My results show different retrieval methods have complementary strengths:

- BM25 performed consistently (identical answers for top_k=5 and 10) with high confidence for certain questions (0.646 for "predictive models")
- Dense retrieval showed more variation and different strengths (0.729 confidence for the drug discovery question)

I would combine both approaches as shown in my implementation:

 # Hybrid approach combining strengths of both methods
bm25_results = retrieve_bm25(query, top_k)
dense_results = retrieve_dense(query, top_k)
combined_results = merge_results(bm25_results, dense_results)

Data Augmentation Techniques
With limited dataset, I would, maximize what I have:

- Generate synthetic question variations for existing documents
- Use my confidence score evaluation to identify successful retrievals (like the 0.646 confidence example)1 and create similar training pairs
- Implement back-translation to create paraphrases of existing questions
- Use a language model to generate additional queries for each document

Few-Shot Learning Approach
Looking at my assignment results where retrieval performance varied by question type:

- Begin with BM25 (which requires no training) as a baseline
- Use metric-based few-shot learning for the dense retriever
- Implement contrastive learning to better distinguish relevant from irrelevant contexts

Iterative Fine-tuning Process
My evaluation framework provides a perfect structure for iterative improvement:

1. Start with baseline retrieval methods (BM25 and pre-trained dense)
2. Analyze performance gaps (like the confidence differences between methods)
3. Focus training on weak areas (questions with low confidence scores)
4. Re-evaluate using the same metrics (token_length, contains_answer, confidence_score)
5. Repeat with adjusted parameters

Cross-Domain Knowledge Transfer
Since my implementation shows performance varies by query type (metabolics study vs. drug discovery), I would:

- Pre-train on a larger related dataset first
- Fine-tune on my domain-specific small dataset
- Use regularization techniques to prevent overfitting

The evaluation methodology in my code (comparing retrieval methods across multiple metrics) provides a solid framework for measuring progressive improvements, even with limited training data.

3. What are some failure cases where a RAG model might underperform compared to a standard Transformer?

Based on the RAG implementation for healthcare papers and its evaluation results, several patterns emerge that highlight when RAG systems might underperform compared to standard Transformer models:

Inconsistent or Contradictory Retrieval Results
The assignment shows significant answer variations for the same question. For "What model is used for metabolics study?", different retrieval methods produced three different answers: "AI", "predictive models", and "Biomarker". When retrieval results contain contradictory information, the final generation may be inconsistent or confused compared to a standard Transformer's coherent (though potentially incorrect) response.

Low Confidence on Specialized Topics
For specialized queries, the RAG system showed extremely low confidence scores. The dense retrieval with top_k=5 produced a confidence score of only 0.008 for "predictive models" when asked about metabolics studies. Standard Transformers might maintain higher confidence on niche topics where retrieval struggles to find relevant documents.

Retrieval Latency Issues
The implementation requires fetching documents, computing embeddings, and performing retrieval before generation - a multi-step process with inherent latency. The code shows both BM25 and dense retrieval operations that add processing time. Standard Transformers generate responses in a single forward pass without this overhead.

Knowledge Recency Problems
When the retrieval corpus becomes outdated (the implementation uses a fixed collection of 10 PubMed papers), RAG models might provide obsolete information. Standard Transformers are limited to their training data cutoff, but don't falsely suggest they're retrieving up-to-date information.

Uninformative Circular Responses
The results show answers like "artificial intelligence" to the question "How does AI help in drug discovery?". This circular response occurs when retrieval finds documents mentioning the query terms but not actually answering the question - a problem standard Transformers might avoid through broader parametric knowledge.

Query-Format Sensitivity
The assignment reveals sensitivity to question formulation. Adding just a question mark changed the confidence score from 0.181 to 0.089 for patient outcomes questions. Standard Transformers might be more robust to minor query variations.

Complex Reasoning Requirements
For questions requiring multi-step reasoning beyond retrieved facts, the RAG model shows limitations. The answer "streamlining logistics and inventory management" for drug discovery questions demonstrates shallow connections rather than deep causal

understanding. Standard Transformers might better leverage their parametric knowledge for complex reasoning.

Misleading Retrieval
For "benefits of machine learning", one configuration retrieved information suggesting it "negatively affects patient outcomes" - directly contradicting the question. When retrieval pulls tangentially related but misleading information, RAG can produce worse answers than standard Transformers.

These failure cases highlight that while RAG excels at knowledge-intensive tasks with reliable retrieval, standard Transformers may perform better when facing specialized queries, requiring complex reasoning, or when retrieval quality is compromised.

Task 4 - Roberta

Conclusion and Observations

This study explores the fine-tuning of RoBERTa for multi-label classification in highly imbalanced datasets. The results show that batch size and learning rate are critical factors in determining model performance. The optimization landscape has distinct basins of attraction, with only specific configurations achieving meaningful convergence. Extreme class imbalance, where 92.5% of the data belongs to a single class, significantly affects performance, requiring post-hoc threshold tuning to improve the detection of minority classes.

Key Observations

1. **Batch Size and Learning Rate Interactions**
    - The best-performing configuration used batch size 64 with a learning rate of 3e-5, achieving the highest micro-F1 score of 0.8509.
    - Smaller batch sizes (16 and 32) resulted in unstable training and poor generalization.
    - Some configurations failed to converge or remained trapped in local minima.
2. **Training Dynamics and Optimization Behavior**
    - The best model exhibited a 25.05% reduction in loss over three epochs, while others showed minimal improvement.
    - Larger batch sizes improved gradient stability, reducing loss variance by 59.7% compared to smaller batches.
3. **Class Imbalance Impact and Threshold Tuning**
    - The initial model failed to detect minority classes, with zero F1-scores for most of them.
    - Threshold tuning improved macro-F1 by 52.2%, increasing recall for rare labels at the cost of precision.
    - The gap between micro-F1 (0.8509) and macro-F1 (0.1203) highlights the challenge of imbalanced classification.
4. **Evaluation Metrics Before and After Threshold Tuning**
    - Micro-Precision: 0.9275 → 0.1806 (-80.5%)
    - Micro-Recall: 0.7859 → 0.9458 (+20.3%)
    - Micro-F1: 0.8509 → 0.3032 (-64.4%)
    - Macro-Precision: 0.1160 → 0.1497 (+29.1%)
    - Macro-Recall: 0.1250 → 0.8003 (+540.2%)
    - Macro-F1: 0.1203 → 0.1831 (+52.2%)
    - ROC-AUC (micro): 0.9581
    - ROC-AUC (macro): 0.8075
    - Mean Average Precision: 0.2211
5. **Class-Specific Performance Analysis Before Optimization**
    - The healthy class had an F1 score of 0.9625 with 100% recall, showing that the model overwhelmingly predicted the majority class.
    - The minority classes had an F1 score of 0.0000, despite ROC-AUC scores ranging from 0.69 to 0.86, suggesting that while the model ranked examples correctly, it struggled with classification.
6. **Computational Efficiency and Training Time**

- Batch size 64 was the most efficient, processing each example 19% faster than batch size 16.
- Larger batch sizes led to lower variance and more stable training but required more memory.

Key Takeaways and Recommendations

1. **Optimal Hyperparameters**
   - Batch size 64 with a learning rate of 3e-5 is the best setting for multi-label classification with RoBERTa.
   - Smaller batch sizes result in unstable gradients and poor optimization.
2. **Addressing Class Imbalance**
   - Post-hoc threshold tuning improves minority class detection but reduces precision.
   - Ranking metrics such as ROC-AUC and PR-AUC provide a better evaluation of model performance in imbalanced datasets.
3. **Efficiency Considerations**
   - Using the largest possible batch size improves stability and computational efficiency.
   - Adaptive learning rate schedules help stabilize training for large batches.
4. **Future Directions for Improvement**
   - Advanced sampling strategies such as focal loss, balanced-batch sampling, or dynamic sampling.
   - Architectural enhancements like label-correlation-aware models and hierarchical attention mechanisms.
   - Ensemble methods combining models trained with different hyperparameters for robustness.
   - Hyperparameter optimization techniques like Bayesian optimization or multi-fidelity tuning.

Final Conclusion

The study provides insights into fine-tuning RoBERTa for multi-label classification in imbalanced datasets, demonstrating that batch size and learning rate selection play a critical role in optimization success. Threshold tuning significantly improves minority class detection, but further improvements may require architectural modifications or advanced sampling strategies.

1. Explain the trade-off between batch size, learning rate, and training stability when fine-tuning large language models.

When fine-tuning large language models such as RoBERTa for multi-label classification tasks, there exists a critical interplay between **batch size**, **learning rate**, and **training stability**. Understanding these relationships is crucial, as they strongly influence model performance, convergence behavior, and generalization. The trade-offs among these hyperparameters are detailed below:

1. Batch Size: Impact on Stability and Convergence
Batch size affects how accurately gradient updates represent the overall dataset:

- **Small Batch Sizes (e.g., 16)**:
    - **Pros:**
        - Lower GPU memory usage allows faster iterations.
        - Facilitates quick, exploratory experimentation.
    - **Cons:**
        - High gradient noise leads to unstable gradient estimates, causing optimization instability.
        - Difficulty escaping poor local minima, resulting in minimal convergence.
    - **Assignment Result:**
        - At batch size 16 and LR 1e-5 to 5e-5, loss values remained essentially unchanged (~0% reduction), indicating inadequate optimization and instability.
- **Large Batch Sizes (e.g., 64):**
    - **Pros:**
        - Reduced gradient noise leads to stable gradient estimates.
        - Enables smoother convergence and effective optimization across epochs.
    - **Cons:**
        - Higher GPU memory requirements.
        - Risk of convergence into sharper minima affecting generalization negatively if excessively large.
    - **Assignment Result:**
        - Batch size 64 and learning rate 3e-5 showed exceptional stability and convergence, reducing training loss substantially (initial loss: 0.1884, final loss: 0.1412, a 25.05% reduction) and achieving optimal performance (Micro-F1 score of 0.8509).

2. Interaction of Learning Rate and Batch Size
The effectiveness of the learning rate varies significantly with batch size:

Batch Size	Optimal Learning Rate	Observed Performance (Micro-F1)
16	High (5e-5)	Poor (~0.30), but marginally better than lower LRs
32	High (5e-5)	Improved (0.6216), indicating better stability
64	Moderate (3e-5)	Exceptional (0.8509), clear optimal point

Explanation:

- Small batch sizes benefit slightly from higher learning rates (5e-5) to help escape local minima, though optimization still remains suboptimal due to inherent instability.
- Medium batch sizes showed improved training behaviour, significantly stabilizing gradient estimates and reducing loss (from 0.6283 to 0.5950).
- Large batch sizes exhibited a complex, non-linear sweet spot (LR=3e-5 at batch size 64), achieving dramatically better performance than alternative learning rates.

3. Training Stability Explained Through Optimization Dynamics

Training stability results from carefully balancing batch size and learning rate:

- **Gradient Stability:**
  - Larger batch sizes provide consistently more reliable gradient updates, resultd by lower batch-level loss variability (Coefficient of Variation from 2.78% at batch size 16 to 1.12% at batch size 64).
- **Optimization Dynamics:**
  - Optimal hyperparameter combinations (batch size 64 and LR=3e-5) produce stable training dynamics, demonstrating significant epoch-over-epoch loss improvements (loss decreased by ~25%).
  - Suboptimal combinations either stagnated or showed unstable loss trajectories, failing to converge effectively.

3. Practical Recommendations

Drawing from the assignment results, the following recommendations emerge:

- **Batch Size:**
  - Choose the largest feasible batch size (ideally 64) for stable gradient estimation and efficient optimization.
- **Learning Rate:**
  - For large batch sizes, prefer moderate learning rates (~3e-5).
  - For smaller batch sizes, consider higher learning rates (5e-5) despite the potential instability.
- **Early Training Monitoring:**
  - Monitor initial training loss closely. Lower initial losses typically indicate stable optimization, whereas high initial losses rarely improve substantially in later epochs.

4. Conclusion

Effectively fine-tuning transformer-based models requires carefully balancing batch size and learning rate to maintain training stability and optimize performance. Larger batch sizes combined with carefully selected moderate learning rates significantly enhance convergence quality, training stability, and overall model effectiveness. Understanding and navigating these trade-offs is essential, especially for challenging tasks like imbalanced multi-label text classification.

2. What are catastrophic forgetting and overfitting, and how can you detect and mitigate them in fine-tuning?

Catastrophic forgetting occurs when a pre-trained language model loses previously acquired knowledge as it adapts to a new task. In the context of transformer models like RoBERTa, this means that the general language understanding capabilities gained during pre-training on large corpora are overwritten or corrupted during fine-tuning on specific downstream tasks.

This phenomenon happens because neural networks update their parameters globally. As the model optimizes for the new task, it modifies weights that were crucial for general language understanding but might be less important for the specific fine-tuning task.

Detecting Catastrophic Forgetting
The experiment results provide several indicators that can help identify catastrophic forgetting:

1. **Initialization-dependent performance variations**: The dramatic performance differences across configurations suggest potential forgetting. For example, the BS=64/LR=3e-5 configuration achieved dramatically better results (F1=0.8509) than BS=64/LR=5e-5 (F1=0.0680).
2. **Early loss dynamics**: As noted in the analysis, initial loss values (0.1884 vs. 0.7582 for different LRs with BS=64) indicates that optimization trajectory is determined extremely early in training. Sharp early loss increases may indicate the model rapidly forgetting pre-trained knowledge.
3. **Basin bifurcation patterns**: The observation that optimization landscape appears to contain two distinct basins of attraction - a suboptimal basin with high loss (~0.6-0.76) and an optimal basin with low loss (~0.14-0.19) suggests models trapped in the high-loss basin may have undergone catastrophic forgetting.
4. **Representation quality degradation**: While not directly measured in the experiment, degradation in ROC-AUC scores (which measure ranking capability) compared to classification metrics can indicate forgetting of general semantic understanding.

Overfitting occurs when a model learns patterns specific to the training data that fail to generalize to unseen examples. For language models, this often manifests as memorizing training examples rather than learning underlying patterns.

Detecting Overfitting
The experiment results demonstrate several methods to identify overfitting:

1. **Training-validation divergence**: Monitoring the gap between training loss and validation metrics. While most configurations didn't show overfitting, this would appear as decreasing training loss with stagnating or declining validation metrics.
2. **Loss trajectory analysis**: The report examined consistency of loss reduction across epochs, noting that only BS=64/LR=3e-5 showed substantial improvement (25.05% loss reduction) without validation performance degradation, indicating it avoided overfitting.
3. **Per-epoch performance tracking**: The experiment tracked metrics across epochs, which helps detect when validation performance plateaus or declines while training continues to improve.

4. **Class-specific performance analysis**: The detailed per-class metrics revealed that while overall micro-F1 was high (0.8509), macro-F1 was much lower (0.1203), suggesting potential overfitting to majority patterns.

Mitigating Catastrophic Forgetting

Based on the assignment findings and general best practices:

1. **Appropriate learning rate selection**: The Assignment results demonstrated that finding the right learning rate is critical. For RoBERTa, a moderate learning rate (3e-5) with larger batch sizes allowed learning new tasks without dramatic forgetting.
2. **Gradual unfreezing**: As suggested in the recommendations, consider gradual unfreezing of transformer layers to allow higher-level representations to adapt before fine-tuning lower layers that contain more general language knowledge.
3. **Learning rate warmup**: The analysis recommends learning rate warmup could help higher learning rates, enabling more stable early training that preserves pre-trained knowledge.
4. **Larger batch sizes**: The experiment showed that larger batch sizes (64) provided more reliable gradient estimates, which helps prevent destructive weight updates that could cause forgetting.
5. **Layer-wise learning rates**: Using lower learning rates for earlier layers (not implemented in the experiment) can help preserve general language knowledge while allowing task-specific adaptation in later layers.

Mitigating Overfitting

The assignment demonstrates several effective strategies:

1. **Early stopping**: The analysis suggests short training runs (1-2 epochs) with different hyperparameters are more valuable than long runs with suboptimal parameters, indicating that limiting training duration prevents overfitting.
2. **Regularization techniques**: The model inherently includes dropout from RoBERTa's configuration, which helps prevent overfitting by randomly disabling neurons during training.
3. **Appropriate batch size**: Larger batch sizes (64) provided more stable training and better generalization than smaller batch sizes (16 or 32).
4. **Gradient clipping**: The experiment employed max_grad_norm=1.0 throughout training, preventing extreme weight updates that could lead to overfitting specific patterns.
5. **Metric selection for model evaluation**: Using multiple evaluation metrics (F1, ROC-AUC, precision, recall) provides a more complete picture of generalization capability than relying on a single metric.

Conclusion

The assignment results support a theoretical understanding where catastrophic forgetting and overfitting represent different failure modes in optimization:

- **Catastrophic forgetting** appears when the model falls into a different basin of attraction than the one containing the pre-trained knowledge.
- **Overfitting** occurs when the model over-optimizes for the training data distribution, moving too far into a narrow basin specific to the training examples.

The remarkable performance of the BS=64/LR=3e-5 configuration suggests it found an optimal path that neither forgot pre-trained knowledge nor overfitted to the training data, resultd by consistently strong performance across both training and evaluation metrics.

Addressing catastrophic forgetting and overfitting during fine-tuning is essential to maintaining model effectiveness and generalization capability. By carefully selecting training strategies, monitoring model performance, and employing targeted mitigation techniques, practitioners can effectively navigate these challenges to achieve optimal fine-tuning outcomes.

3. Given an imbalanced dataset, what strategies would you use to ensure fair evaluation and training?

When working with imbalanced datasets, standard training and evaluation approaches can lead to models that perform well on majority classes but poorly on minority classes. The search results demonstrate this challenge with the healthy class comprising 92.5% of samples while minority classes represent as little as 2.0%. To ensure fairness, several strategies should be employed:

Data-Level Strategies

1. Resampling Techniques

- **Oversampling**: Increase the number of minority class samples through duplication or synthetic sample generation
- **Undersampling**: Reduce majority class samples to create more balanced training distributions
- **Hybrid approaches**: Combine both strategies to achieve optimal balance

2. Data Augmentation

- Generate synthetic examples of minority classes to address class imbalance
- For text data, techniques like back-translation, synonym replacement, or context-preserving perturbations can create valid variations

3. Stratified Sampling

- Maintain class distributions across train/validation/test splits
- Ensure rare classes appear in all splits proportionally

Algorithm-Level Strategies

1. Loss Function Modifications

- **Class weighting**: Apply inversely proportional weights to class frequencies in the loss function
- **Focal Loss**: Modify standard BCE loss to focus more on hard examples and minority classes
- **Asymmetric Loss**: Apply different weights to positive and negative examples for each class

2. Learning Parameters Optimization

- **Batch size selection**: As demonstrated in the results, larger batch sizes (64) improved minority class detection by providing more examples per batch

- **Learning rate calibration**: Finding the optimal learning rate (3e-5) significantly improved model performance on minority classes

3. Model Architecture Adjustments

- Implement hierarchical attention mechanisms or multi-task learning approaches that might better capture minority class patterns
- Consider ensemble methods combining models trained with different hyperparameters or random seeds

Evaluation Strategies

1. Multiple Metric Assessment

- **Beyond accuracy**: Use metrics like F1-score, precision, recall that better reflect performance on minority classes
- **Macro vs. micro averaging**: The search results show a dramatic difference between micro-F1 (0.8509) and macro-F1 (0.1203), directly quantifying imbalance effects
- **Per-class metrics**: Analyze performance for each class individually rather than only looking at aggregate metrics

2. Ranking vs. Classification Metrics

- **ROC-AUC and PR-AUC**: The experiment showed high ROC-AUC scores (0.69-0.86) despite poor F1-scores (0.00), indicating good ranking ability despite classification challenges
- **Average precision**: Provides insights into model's ranking capability per class

3. Confusion Matrix Analysis

- Examine true/false positives and negatives for each class to understand error patterns
- The experiment revealed how minority classes suffered from high false positives after threshold optimization

Post-Processing Techniques

1. Threshold Optimization

- **Class-specific thresholds**: The search results demonstrated substantial improvements (macro-F1 +52.2%) by optimizing decision thresholds per class
- The optimal thresholds varied dramatically (0.00-0.55) based on class prevalence

2. Calibration Techniques

- Apply Platt scaling or isotonic regression to calibrate probabilities
- Optimize for specific performance metrics based on application needs

3. Cost-Sensitive Decision Making

- Apply different costs to different types of errors based on class importance
- Prioritize recall or precision depending on the domain requirements

Conclusion

The experiment in the search results implemented several of these strategies:

1. Explored optimal hyperparameters that improved minority class detection
2. Applied post-hoc threshold tuning that improved macro-F1 by 52.2%
3. Evaluated using multiple metrics (F1, ROC-AUC, precision, recall) across different averaging methods
4. Analyzed per-class confusion matrices to understand error patterns

By combining these data-level, algorithm-level, evaluation, and post-processing strategies, you can develop models that perform fairly across all classes despite significant imbalance in the training data.

Handling class imbalance effectively is crucial for fair model training and evaluation. By leveraging resampling techniques, class-weighted loss functions, and comprehensive evaluation metrics, practitioners can build models that generalize well across all classes, preventing biases toward dominant categories. Ensuring fair evaluation requires going beyond standard accuracy metrics and considering the impact of imbalance on real-world deployment.