English–Dutch Machine Translation Using mT5 and LoRA

1. Introduction

This project focuses on building and fine-tuning a **Neural Machine Translation (NMT)** system for **English to Dutch translation**. A pretrained multilingual Transformer model (**mT5-base**) is adapted using **Parameter-Efficient Fine-Tuning (LoRA)** to achieve high-quality translations with reduced computational cost.

---

2. Environment Setup & Dependencies

The project was implemented in a Python-based Jupyter/Google Colab environment. Required libraries were installed using `pip`.

Key Libraries: - PyTorch, TorchVision - PyTorch Lightning - Hugging Face Transformers - PEFT (LoRA) - Datasets, SacreBLEU - Pandas, OpenPyXL

GPU availability was verified using `nvidia-smi`.

---

3. Data Collection & Preprocessing

3.1 Dataset Sources

- **Europarl v7** English–Dutch parallel corpus
- Additional English text source for alignment
- External evaluation datasets:
- Challenge Dataset (Excel format)
- FLORES Dev-Test EN–NL benchmark

3.2 Data Loading & Inspection

Source files were inspected for encoding and structure issues. Appropriate encodings (`utf-8`, `latin-1`) were used to avoid parsing errors.

3.3 Sentence Alignment

English and Dutch sentence lists were truncated to the same length to ensure one-to-one alignment.

3.4 DataFrame Construction

A pandas DataFrame was created with standardized columns: - `English Source` - `Reference Translation`

This structured format supports downstream model training and evaluation.

4. Dataset & Data Handling

A custom PyTorch `Dataset` class (`TranslationDataset`) was implemented to: - Format translation prompts - Tokenize source and target sentences - Apply padding and truncation - Mask padding tokens for loss computation

The dataset is compatible with Transformer-based sequence-to-sequence models and PyTorch Lightning.

---

5. Model Selection & Configuration

5.1 Pretrained Model

- **Model:** `google/mt5-base`
- A multilingual sequence-to-sequence Transformer suitable for translation tasks

5.2 Tokenizer Setup

The tokenizer was loaded from the pretrained model and configured with EOS tokens for padding.

5.3 LoRA Fine-Tuning

Low-Rank Adaptation (LoRA) was applied to attention **query (q)** and **value (v)** layers to reduce the number of trainable parameters while maintaining performance.

---

6. Training Data Pipeline

The training pipeline consists of: - Custom translation dataset initialization - PyTorch `DataLoader` with shuffling enabled - Small batch size to accommodate GPU memory constraints

This setup ensures efficient and scalable data feeding during training.

---

7. Model Training Framework

7.1 PyTorch Lightning Module

A custom Lightning module (`TranslationFineTuner`) encapsulates: - Forward pass and loss computation - Training step logic - Optimizer configuration (AdamW)

7.2 Training Configuration

- Epochs: 3
- Precision: Mixed precision (FP16)
- Gradient clipping enabled
- Automatic device and accelerator selection

This configuration provides stable and efficient fine-tuning.

---

8. Inference & Translation Pipeline

An inference function was implemented to: - Switch the model to evaluation mode - Generate translations using beam search - Decode outputs into readable Dutch text

Beam search improves translation quality, while device-aware execution ensures GPU compatibility.

---

9. Model Evaluation Metrics

Translation quality was evaluated using standardized corpus-level metrics: - **BLEU** – N-gram precision-based metric - **chrF++** – Character-level F-score - **TER** – Translation Edit Rate

The `sacrebleu` library ensures reproducible and benchmark-comparable results.

---

10. Software-Domain Evaluation

A random sample of 100 sentence pairs was selected for domain-specific evaluation.

Evaluation Steps:

1. Sample selection with fixed random seed
2. Translation generation using the fine-tuned model
3. Metric computation against reference translations

This approach provides a quantitative assessment of performance on software-related text.

---

11. Results & Discussion

The evaluation metrics demonstrate that: - The LoRA-adapted mT5 model produces coherent and contextually accurate Dutch translations - chrF++ scores indicate strong character-level alignment - TER values suggest reduced post-editing effort

Overall, parameter-efficient fine-tuning achieved strong performance with limited computational resources.

---

12. Conclusion

This project successfully implemented an **English–Dutch machine translation system** using: - A pretrained multilingual Transformer (mT5) - Efficient LoRA-based fine-tuning - Robust training, inference, and evaluation pipelines

The solution is scalable, resource-efficient, and suitable for real-world deployment.

---

13. Future Enhancements

- Expand training data with additional parallel corpora
- Domain-adaptive fine-tuning for technical or legal text
- Hyperparameter optimization
- Model deployment as an API or web service

---

14. References

- Hugging Face Transformers Documentation
- PyTorch & PyTorch Lightning Documentation
- SacreBLEU Evaluation Toolkit
- FLORES Multilingual Benchmark