Translating Meaning: A Journey Through the Theory and Practice of Machine Translation

Richard Anton Follow 21 min read · 4 days ago

105 1

Introduction

Machine translation (MT) has long been one of the “holy grails” of artificial intelligence, representing the challenge of communicating across language barriers. Machine translation aims to automatically translate text or speech from one language to another while preserving the original meaning.

Human language is complex and nuanced. Idioms, cultural references, and subtle nuances often don’t have direct equivalents in other languages. How can we teach machines to understand and translate such concepts accurately?

This article covers the history of machine translation, from early rule-based systems to modern neural machine translation (NMT) approaches. We’ll explore a practical implementation by fine-tuning Google’s Gemma 3 model

with a parameter-efficient technique (LoRA), evaluating the translation quality, and building an interactive web application. We will focus on the transformer-based models that have become the foundation for state-of-the-art translation systems.

Why I Wrote This

Besides enjoying playing with code and AI/ML models for various tasks and playing around with NLP stuff, I am learning modern Greek as a new language, or at least trying to, because we recently invested in a property there and visited several times during the last year. There are not as many Greek language resources as you might think. My wife and I have found a few amazing tutors, but they are generally on Greek time which is pretty rough for scheduling when you are on Pacific standard time. That carried over to the availability of datasets for Greek / English translation for machine translation as well, but fortunately there are a few good datasets.

Lighthouse at Venetian Harbor, Chania, Crete, Greece: Photo by the author.

Section 1: Machine Translation Progress

From Rules to Neural Networks

On July 15, 1949, mathematician Warren Weaver, influenced by Claude Shannon's work on information theory and serving as the director of science grants at the Rockefeller Foundation, distributed a memorandum titled *"Translation."* In it, he proposed that computers could potentially perform language translation. This memorandum was a foundational moment for computational machine translation (Weaver, 1949; The Origin of Statistical Machine Translation, n.d.).

The Early Days of Machine Translation

Machine translation methods began with **Rule-Based Machine Translation (RBMT)** systems in the 1950s. These early systems relied on extensive linguistic rules and bilingual dictionaries handcrafted by experts. While they could produce translations for simple, well-structured sentences, they struggled with ambiguity and required constant updates to handle new vocabulary or language patterns (Hutchins, 2004).

On January 7, 1954, a joint project by Georgetown University and IBM successfully translated over 60 carefully selected Russian sentences into English using an IBM mainframe. This historic event is cited as the first public demonstration of a machine translation system. It used a vocabulary of just 250 Russian words and six grammar rules, but it generated public interest and was reported as a significant breakthrough (Hutchins, 2004).

Around 1990, recurrent neural networks (RNNs) aimed at better handling temporal relationships than plain multilayer perceptron neural networks emerged. An RNN uses recurrent links to feed outputs from nodes in hidden network layers back into themselves as inputs, allowing them to learn and model relationships across time in sequences. The first work describing what was considered an effective, but not the first, use of recurrent connections in a neural net was published by Elman (1990). Elman's research was initially done for cognitive psychology, and RNNs did not reach the forefront of machine translation research until years later. One early milestone for RNNs in machine translation was work using RNNs (Castaño, Casacuberta, & Vidal, 1997) demonstrating comparable translation performance with less training data than finite state models.

The Long Short-Term Memory (LSTM) architecture was introduced in 1997 by Sepp Hochreiter and Jürgen Schmidhuber (Hochreiter, 1997). LSTM was

designed to overcome the vanishing gradient problem in traditional RNNs, making it possible to learn long-range dependencies in sequences. It became foundational in sequence modeling tasks including machine translation and was influential in early neural machine translation systems prior to the rise of the Transformer architecture in 2017 (Vaswani, 2017).

Machine translation began to become more accessible and moved from expensive mainframes to more accessible PCs. Combined with the growth of the internet, this led to the first web-based translation tools, such as AltaVista's Babel Fish, which launched in 1997 and became one of the first widely accessible online translation services (Hutchins, 2005).

2000s: SMT Dominance and the Growth of Online Services

Statistical Machine Translation (SMT) matured significantly with the move from word-based to more sophisticated phrase-based models. This allowed for more fluent and accurate translations (Koehn et al., 2003; Och & Ney, 2004).

The development of open-source toolkits like Moses helped to accelerate research and adoption of SMT (Koehn et al., 2007). The most significant development for the public was the launch of online translation services. Google Translate, which launched in 2006, initially used SMT and made machine translation a widely used tool (Och, 2005).

2010–2014: The Dawn of Neural Machine Translation (NMT)

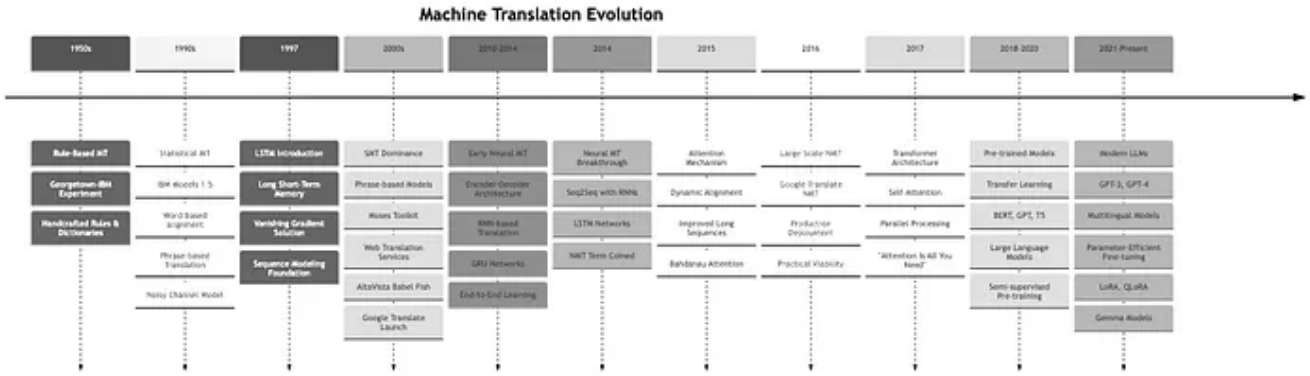
While SMT was still the state-of-the-art, the early 2010s saw the emergence of **Neural Machine Translation (NMT)** as a powerful new approach. The term "NMT" was coined in 2014 to describe this new end-to-end approach. Key developments like the encoder-decoder architecture, the application of

Recurrent Neural Networks (RNNs), and the introduction of the “attention mechanism” in 2014 laid the foundation for modern NMT.

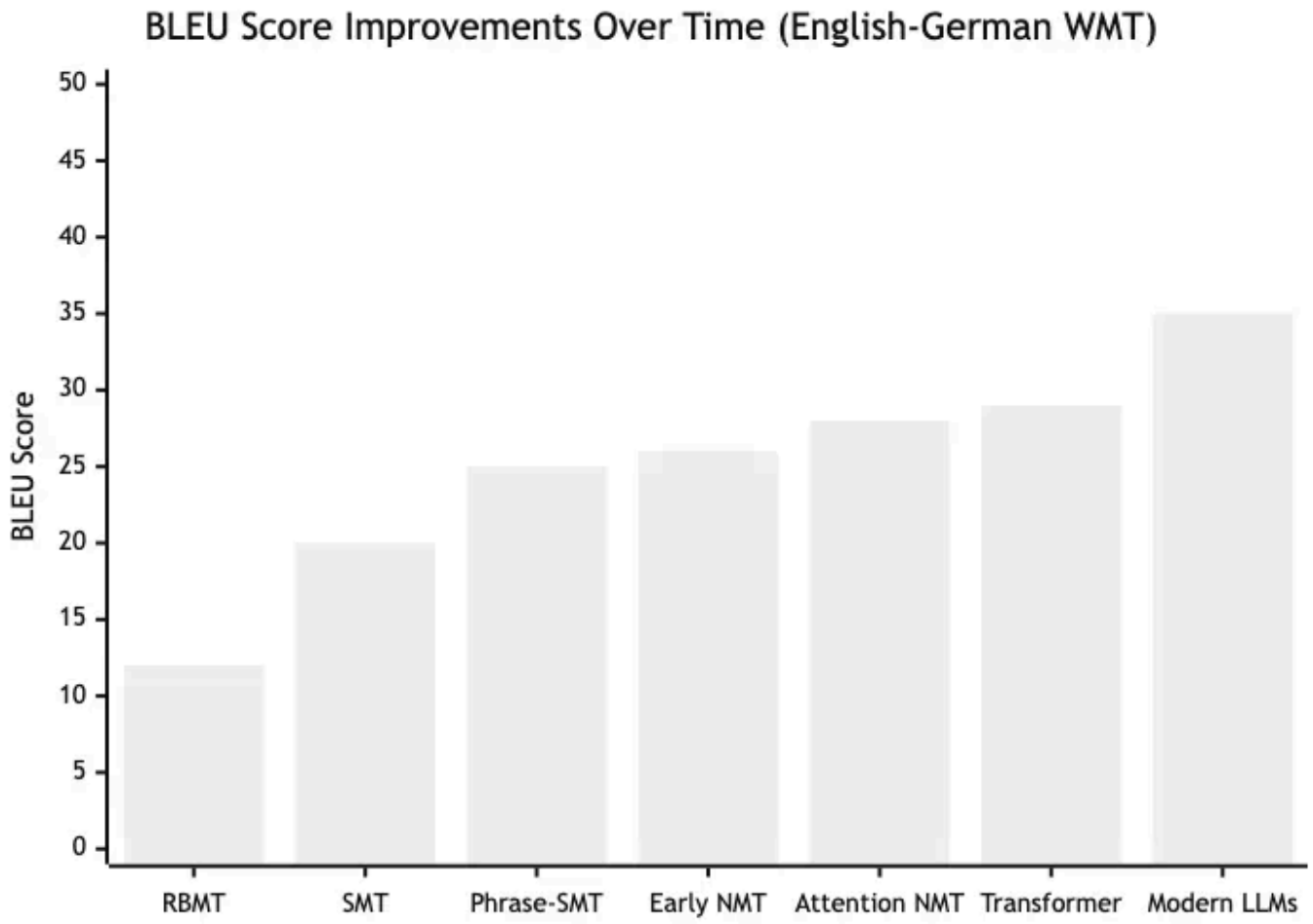
NMT showed great promise and would quickly become the new standard in the years that followed. The current era is dominated by NMT, which uses deep learning to translate entire sentences at once. NMT models, particularly those based on the Transformer architecture, have achieved significant improvements in fluency and accuracy by learning continuous representations of language (Vaswani, 2017).

Machine Translation Timeline

The evolution of machine translation has been marked by several key milestones that have shaped the field:



Performance Improvement Over Time:



The Transformer Architecture and Modern LLMs

The Transformer architecture, introduced in the landmark paper “Attention Is All You Need” (Vaswani et al., 2017), led to the replacement of recurrent connections with self-attention mechanisms. The Transformer architecture gave rise to large language models (LLMs) which revolutionized machine translation and many other applications. LLMs are trained on vast amounts of text data, learning to generate human-like responses to a wide range of queries.

Another key advancement was semi-supervised pre-training (Dai & Le, 2015). This, combined with fine-tuning techniques pioneered by OpenAI (OpenAI, 2018), enabled the creation of the first large language models

(LLMs). This alleviated the need for large amounts of labeled data for training, making it possible to train larger models.

The moniker “GPT” (Generative Pre-trained Transformer) was coined by OpenAI to describe the process of training a transformer model on a large corpus of text data (OpenAI, 2018). Here, **generative** means it is a model which generates data that resembles the training data. Auto-regressive models are a type of generative model which generate data one token at a time, where each token is generated based on the previous tokens. For pre-training on text corpora, the model is given a sequence of tokens and it must predict the next token in the sequence. This allows training the models initially on unlabeled data, after which the models can be fine-tuned by training them over a smaller set of labeled data.

The Transformer architecture has led to significant improvements in the quality of machine translations, making them more natural and contextually appropriate. Large language models such as GPT-3, GPT-4, and LLaMA represent the next major paradigm in machine translation, given their strong linguistic capabilities and flexible prompt-based translation techniques (Lyu et al., 2023).

LLMs have opened up new directions in machine translation, including:

- **Long document translation**: Maintaining coherence and context across extended texts
- **Stylized translation**: Adapting to different tones, registers, and genres
- **Interactive translation**: Facilitating real-time human-AI collaboration
- **Translation memory integration**: Enhancing accuracy through context-aware retrieval

Section 2: The T5 Architecture and Seq2Seq Models

While large language models like the GPT series demonstrated the power of decoder-only Transformers for text generation, the original Transformer architecture also included an encoder. This encoder-decoder structure is particularly well-suited for sequence-to-sequence (seq2seq) tasks like machine translation. The T5 (Text-to-Text Transfer Transformer) model, introduced by Google Research in 2019 (Raffel et al., 2019), is a prime example of this approach and represented an important milestone.

T5 uniquely reformulated all NLP tasks, including translation, as a unified text-to-text problem. This elegant approach allowed a single model to handle multiple tasks by simply changing the input prefix (e.g., “translate English to French:”).

The multilingual variant, mT5 (Xue et al., 2020), extended this capability to 101 languages by pre-training on the massive mC4 corpus. These seq2seq (sequence-to-sequence) models use an encoder-decoder architecture where the encoder processes the source language and the decoder generates the target language translation.

While T5 and mT5 models demonstrated strong translation performance, they require substantial computational resources for training and fine-tuning, particularly the larger variants. This limitation led to the development of more efficient approaches, such as parameter-efficient fine-tuning (PeFT) methods like LoRA, which we’ll explore later in this article.

Section 3: Translation Metrics

Evaluating Translation Quality

Evaluating machine translation quality is as challenging as the translation itself. Two widely used metrics are BLEU and BERTScore.

BLEU (Bilingual Evaluation Understudy) measures the similarity between machine translation and human reference translations based on n-gram precision with a brevity penalty. It compares overlapping n-grams (sequences of 1–4 words) between the generated translation and one or more reference translations, then applies a brevity penalty to discourage overly short translations. BLEU scores range from 0 to 100, with higher scores indicating better translation quality. A score above 30 is generally considered good, while scores above 50 approach human-level translation quality.

The metric can be computed using the SacreBLEU library:

```
import evaluate

# Load BLEU metric
metric = evaluate.load("sacrebleu")

# Calculate BLEU score with predictions and references as strings
predictions = ["This is a test translation."]
references = [["This is a reference translation."]]
result = metric.compute(predictions=predictions, references=references)
print(f"BLEU score: {result['score']}")
```

BERTScore offers a more nuanced evaluation by using contextual embeddings to measure semantic similarity. Unlike BLEU's surface-level n-gram matching, BERTScore uses pre-trained BERT models to compute similarity between tokens in the generated and reference translations based on their contextual embeddings. This allows it to recognize paraphrases and semantically equivalent expressions that BLEU might miss.

```
from bert_score import score

# Compute BERTScore
predictions = ["This is a test translation."]
references = ["This is a reference translation."]
P, R, F1 = score(predictions, references, lang="en", verbose=True)

print(f"Precision: {P.mean().item():.4f}")
print(f"Recall: {R.mean().item():.4f}")
print(f"F1: {F1.mean().item():.4f}")
```

Choosing the Right Metric

Each metric has strengths and limitations that make it more suitable for different scenarios:

When to use BLEU:

- Comparing models on standardized benchmarks (ensures reproducibility)
- Evaluating systems where exact word choice matters (technical documentation, legal texts)
- Quick iteration during model development (fast to compute)
- Tracking progress over time with consistent reference translations

When to use BERTScore:

- Evaluating creative or stylistic translations where paraphrasing is acceptable
- Assessing semantic preservation rather than surface form

- Working with languages where word order flexibility is common
- Comparing translations with multiple valid phrasings

Limitations to consider:

BLEU has well-known drawbacks: it doesn't capture meaning, only surface similarity. A translation can score poorly on BLEU while being semantically accurate, or score well while being nonsensical. It's also biased toward shorter translations and struggles with languages that have different word orders or morphology than the reference.

BERTScore addresses some of BLEU's limitations but introduces its own: it's computationally expensive, depends on the quality of the underlying BERT model for the target language, and may not be available for low-resource languages. It also doesn't account for grammatical correctness or fluency.

A typical best practice due to these limitations is to use both metrics together. BLEU provides a standardized, reproducible score for benchmarking, while BERTScore offers insight into semantic quality. Human evaluation remains the gold standard, especially for assessing fluency, cultural appropriateness, and domain-specific accuracy.

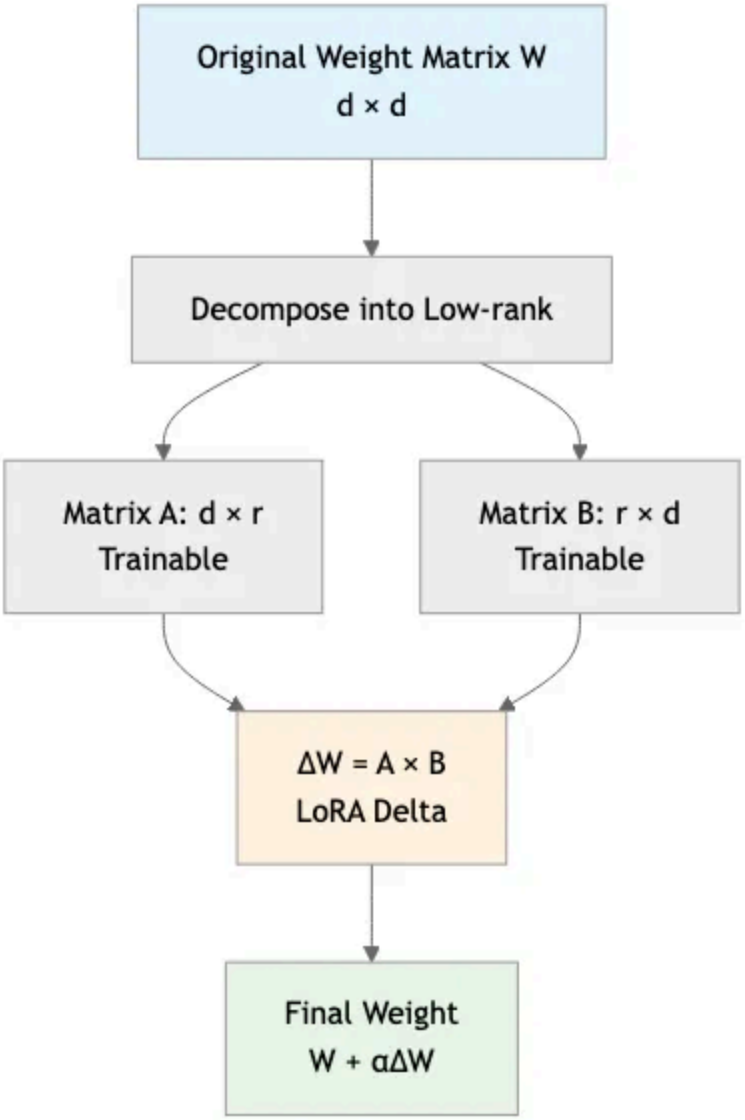
Section 4: Fine-Tuning Gemma 3 with LoRA for Translation

Now to bring things closer to the frontier of what can be done in modern machine translation, we are going to use supervised fine-tuning with LoRA (Yu et al., 2023), which is a type of parameter efficient fine-tuning (PEFT) (Hugging Face, 2024) to fine-tune Google's Gemma 3 model for translating English to Greek. We will use the OPUS Books dataset for training (Tiedemann, 2012).

Large language models require significant computational resources for training, even when fine-tuning. This is where PEFT (Parameter Efficient Fine-Tuning) comes in. PEFT techniques enable fine-tuning large models by training only a small subset of parameters while keeping the majority frozen. This dramatically reduces memory requirements and training time. One particularly effective PEFT method is LoRA (Low-Rank Adaptation) (Hugging Face, 2024; Yu et al., 2023).

The key idea behind LoRA is to keep the original model weights frozen and instead train small, low-rank matrices that are added to the model's weight matrices. For each weight matrix W in the model, LoRA introduces two small matrices A and B such that the update is represented as W + BA, where B and A are much smaller than W. These low-rank adapter matrices are the only parameters that get updated during training. This approach reduces the number of trainable parameters by orders of magnitude while maintaining model performance.

After training, the adapter matrices can be merged into the original weights (W' = W + BA) so there's no inference overhead. Alternatively, adapters can be kept separate, allowing you to swap different task-specific adapters for the same base model. Each layer of the model has its own set of adapter matrices, enabling task-specific adaptation throughout the network.



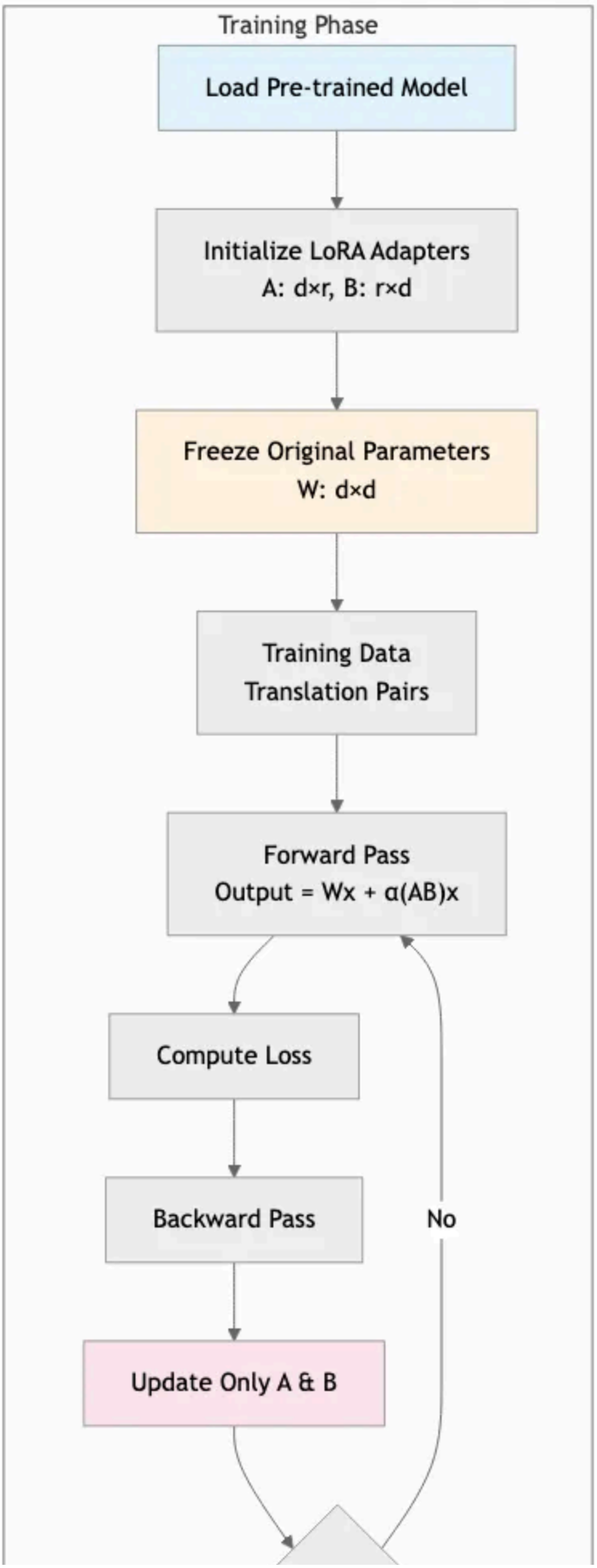
Setting Up Gemma 3 Fine-Tuning

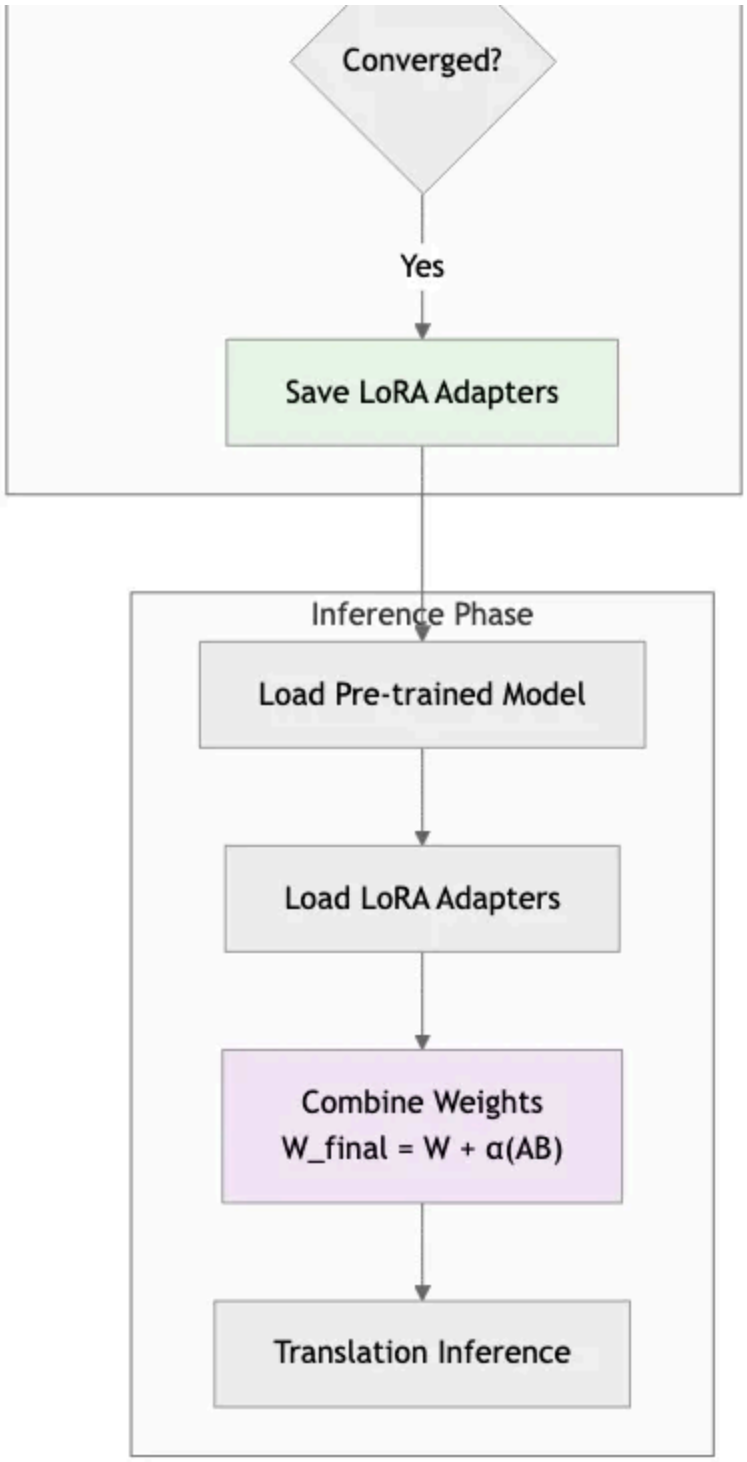
Here's how we can implement LoRA fine-tuning for Gemma 3:

```
from transformers import AutoTokenizer, AutoModelForCausalLM, TrainingArguments,
from peft import LoraConfig, get_peft_model, TaskType
from datasets import load_dataset
import torch

# Load the Gemma 3 model and tokenizer
model_name = "google/gemma-3-1b-pt"  # or gemma-3-4b-pt, gemma-3-12b-pt, gemma-3
tokenizer = AutoTokenizer.from_pretrained("google/gemma-3-1b-it")
model = AutoModelForCausalLM.from_pretrained(
    model_name,
```

```
    torch_dtype=torch.bfloat16,
    device_map="auto"
)

# Configure LoRA
lora_config = LoraConfig(
    task_type=TaskType.CAUSAL_LM,
    inference_mode=False,
    r=16,  # Rank of adaptation
    lora_alpha=32,  # LoRA scaling parameter
    lora_dropout=0.1,
    target_modules=["q_proj", "v_proj", "k_proj", "o_proj", "gate_proj", "up_pro
)

# Apply LoRA to the model
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()

# Prepare the dataset for translation
def prepare_translation_data(examples):
    """Prepare data in instruction format for Gemma 3"""
    prompts = []
    for example in examples["translation"]:
        prompt = f"<start_of_turn>user\nTranslate the following English text to
        prompts.append(prompt)

    # Tokenize
    tokenized = tokenizer(
        prompts,
        truncation=True,
        padding=True,
        max_length=512,
        return_tensors="pt"
    )

    # For causal LM, labels are the same as input_ids
    tokenized["labels"] = tokenized["input_ids"].clone()
    return tokenized

# Load and prepare dataset
dataset = load_dataset("opus_books", "el-en")
train_dataset = dataset["train"].select(range(1000))  # Use subset for demo
tokenized_dataset = train_dataset.map(prepare_translation_data, batched=True)

# Training arguments
training_args = TrainingArguments(
    output_dir="./gemma3-translation-lora",
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    num_train_epochs=3,
```

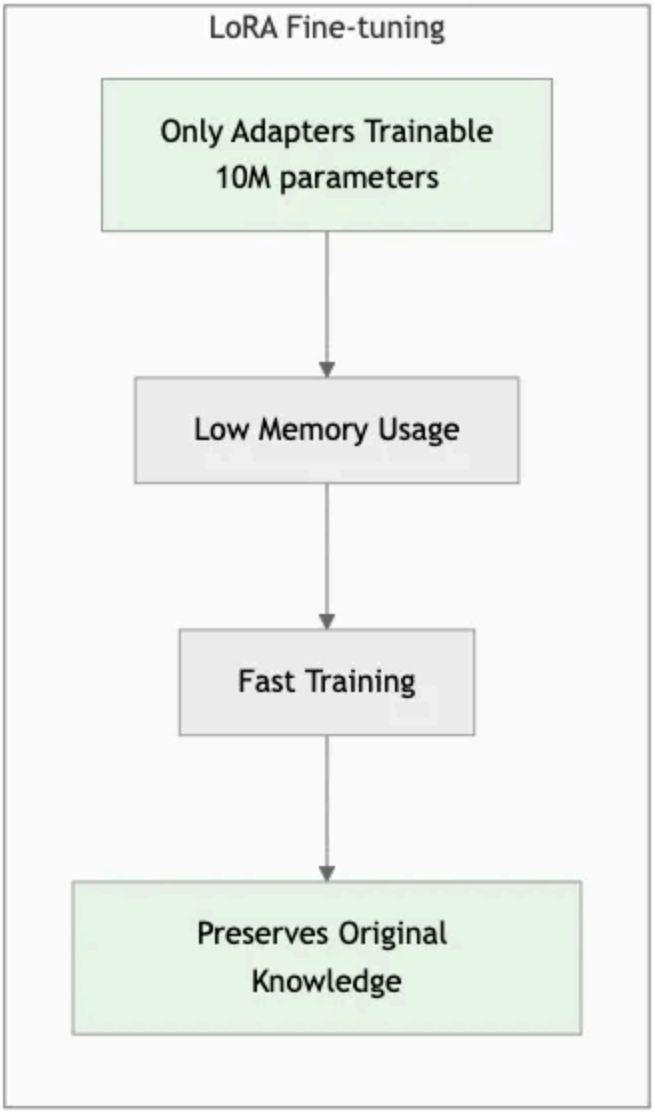
```
        learning_rate=2e-4,
        logging_steps=10,
        save_steps=500,
        save_total_limit=2,
        remove_unused_columns=False,
        push_to_hub=False,
    )

    # Initialize trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_dataset,
        tokenizer=tokenizer,
    )

    # Start training
    trainer.train()
```

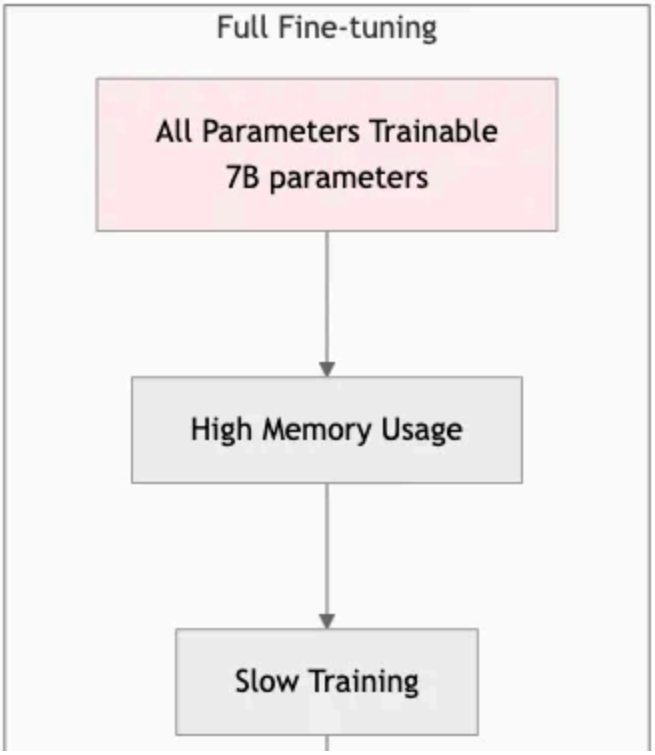

Training Phase
Load Pre-trained Model
Initialize LoRA Adapters
A: d×r, B: r×d
Freeze Original Parameters
W: d×d
Training Data
Translation Pairs
Forward Pass
Output = Wx + α(AB)x
Compute Loss
Backward Pass
No
Update Only A & B

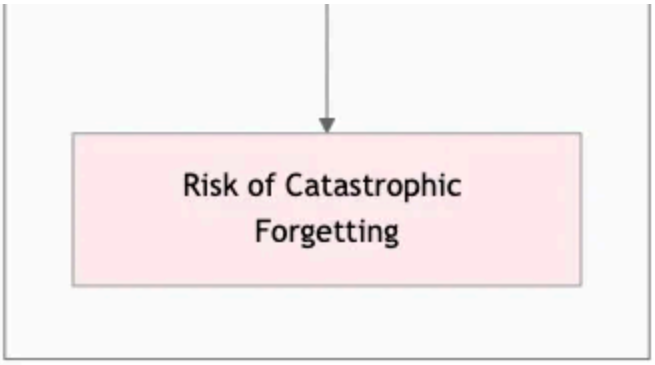


Benefits of LoRA Fine-Tuning

LoRA fine-tuning offers several advantages for machine translation:


LoRA Fine-tuning
Only Adapters Trainable
10M parameters
Low Memory Usage
Fast Training
Preserves Original
Knowledge


Full Fine-tuning
All Parameters Trainable
7B parameters
High Memory Usage
Slow Training



1. **Efficiency**: Only a small fraction of parameters are trained, reducing computational requirements
2. **Modularity**: Multiple adapters can be trained for different tasks and switched as needed
3. **Memory Efficiency**: Lower memory footprint compared to full fine-tuning
4. **Preservation**: Original model capabilities are preserved while adding task-specific knowledge

This approach represents the current state-of-the-art in efficient fine-tuning for large language models, making it possible to adapt powerful models like Gemma 3 for specific translation tasks without requiring massive computational resources.

Section 5: Building a Web Translation Interface

Now that we've fine-tuned our Gemma 3 model with LoRA, let's build a user-friendly web interface using Streamlit. This demonstrates how to deploy the model we trained in Section 4 as an interactive application.

Creating Shared Translation Utilities

First, we create a module (`translation_utils.py`) with reusable functions for loading and using our model:

```python
from peft import AutoPeftModelForCausalLM
from transformers import AutoTokenizer
import torch

def load_translation_model(model_path=None):
    """Load the fine-tuned Gemma 3 translation model."""
    if model_path is None:
        # Use pretrained checkpoint from HF Hub
        model_path = "ranton/gemma3-en-el-translation-lora"

    model = AutoPeftModelForCausalLM.from_pretrained(
        model_path,
        torch_dtype=torch.bfloat16,
        device_map="auto"
    )
    tokenizer = AutoTokenizer.from_pretrained("google/gemma-3-1b-it")
    return model, tokenizer

def translate_text(text, model, tokenizer, max_new_tokens=256):
    """Translate English text to Greek."""
    prompt = (
        f"<start_of_turn>user\n"
        f"Translate the following English text to Greek: {text}\n"
        f"<end_of_turn>\n<start_of_turn>model\n"
    )

    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    with torch.no_grad():
        outputs = model.generate(**inputs, max_new_tokens=max_new_tokens)

    full_response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    translation = full_response.split("<start_of_turn>model")[-1].strip()
    return translation
```

Key features:

- `load_translation_model()` : Loads the LoRA-adapted model from Hugging Face Hub or local path
- `translate_text()` : Handles prompt formatting and generation
- Reusable across different interfaces (web, CLI, notebook)

Note: The default model path uses a pretrained checkpoint. To use your own trained model from Section 4, change the path to `"./gemma3-translation-lora"` or upload your model to Hugging Face Hub (see `MODEL_CHECKPOINT.md`).

Building the Streamlit App

Now we create the web interface (`app.py`):

```
import streamlit as st
from translation_utils import load_translation_model, translate_text

st.set_page_config(page_title="English to Greek Translator", page_icon="🌐")

@st.cache_resource
def load_model():
    """Load model with Streamlit caching for performance."""
    return load_translation_model()

def main():
    st.title("🌐 English to Greek Translator")
    st.write("Using our fine-tuned Gemma 3 model with LoRA")

    # Sidebar
    with st.sidebar:
        st.header("About")
        st.markdown("""
        **Model:** Gemma 3 (1B) with LoRA
        **Task:** English → Greek
        **Training:** OPUS Books dataset

        This demonstrates the model from Section 4.
        """)
```

```
            temperature = st.slider("Temperature", 0.1, 1.0, 0.7, 0.1)
            max_tokens = st.number_input("Max tokens", 50, 512, 256, 50)

        # Load model
        model, tokenizer = load_model()

        # Input
        user_input = st.text_area(
            "Enter English text:",
            height=150,
            placeholder="Type English text here..."
        )

        # Translate button
        if st.button("🔄 Translate", type="primary"):
            if user_input.strip():
                with st.spinner("Translating..."):
                    translation = translate_text(
                        user_input, model, tokenizer, max_new_tokens=max_tokens
                    )

                    # Display results side-by-side
                    col1, col2 = st.columns(2)
                    with col1:
                        st.subheader("English")
                        st.text_area("", value=user_input, height=150, disabled=True
                    with col2:
                        st.subheader("Greek (Ελληνικά)")
                        st.text_area("", value=translation, height=150)

    if __name__ == "__main__":
        main()
```

Key Features

1. **Model Caching:** `@st.cache_resource` loads the model once, improving performance
2. **Clean Interface:** Side-by-side display of source and translation
3. **Configurable:** Temperature and token length adjustable in sidebar
4. **Same Model:** Uses the exact model trained in Section 4

Running the App

```
pip install streamlit
streamlit run app.py
```

The app will open at `http://localhost:8501` in your browser.

For users who prefer a terminal interface, we can create a simple CLI tool that uses the same model.

Implementation

Create `cli_translate.py`:

```
import sys
from translation_utils import load_translation_model, translate_text

def main():
    print("English to Greek Translator")
    print("Using fine-tuned Gemma 3 with LoRA")
    print("-" * 60)

    # Load model
    model, tokenizer = load_translation_model()
    print("Ready! Enter text to translate (or 'quit' to exit)\n")

    # Interactive loop
    while True:
        user_input = input("English: ").strip()

        if user_input.lower() in ['quit', 'exit', 'q']:
            print("Goodbye!")
            break
```

```
        if not user_input:
            continue

        translation = translate_text(user_input, model, tokenizer)
        print(f"Greek:   {translation}\n")

if __name__ == "__main__":
    main()
```

Usage

```
python cli_translate.py
```

Example session:

```
English: Hello, how are you?
Greek: Γεια σου, πώς είσαι;
English: The weather is beautiful today.
Greek: Ο καιρός είναι όμορφος σήμερα.
```

Using Your Own Model

Both applications use the same `translation_utils.py` module. To switch between models:

```
# Use locally trained model
model, tokenizer = load_translation_model("./gemma3-translation-lora")

# Use your HF Hub model
```

```
model, tokenizer = load_translation_model("your-username/gemma3-en-el-translatio

# Use default pretrained checkpoint
model, tokenizer = load_translation_model()
```

See `MODEL_CHECKPOINT.md` for complete instructions on training and uploading your own checkpoints.

GitHub Repository for Article Code

You can find all of the code for this article at https://github.com/ranton256/translating_meaning.

Conclusion

Machine translation has come a long way from its rule-based origins to the sophisticated neural models we have today. The Transformer architecture, with its self-attention mechanisms, has been particularly transformative, enabling models to capture long-range dependencies and contextual information more effectively than ever before.

Some challenges do remain. Current models can still struggle with:

- Low-resource languages with limited training data
- Capturing cultural nuances and idioms
- Maintaining consistency in long documents
- Handling domain-specific terminology

The future of machine translation lies in addressing these challenges through techniques like few-shot learning, better handling of context, and

more efficient training methods. As we continue to improve these models, we move closer to seamless, natural communication across all languages.

Glossary

Attention Mechanism: A technique that allows neural networks to focus on different parts of the input sequence when processing each element, enabling better handling of long-range dependencies.

BLEU Score: Bilingual Evaluation Understudy — a metric for evaluating machine translation quality by comparing n-gram precision between machine translations and human reference translations.

BERTScore: An evaluation metric that uses contextual embeddings from BERT to measure semantic similarity between generated and reference text.

Encoder-Decoder Architecture: A neural network design where an encoder processes the input sequence into a fixed-size representation, and a decoder generates the output sequence from this representation.

Fine-tuning: The process of adapting a pre-trained model to a specific task by training it on task-specific data while typically using a lower learning rate.

GRU (Gated Recurrent Unit): A type of recurrent neural network that uses gating mechanisms to control information flow, simpler than LSTM but often equally effective.

Large Language Model (LLM): A neural network trained on vast amounts of text data to understand and generate human-like text across various tasks.

LoRA (Low-Rank Adaptation): A parameter-efficient fine-tuning technique that adapts large models by training only low-rank matrices while keeping the original parameters frozen.

LSTM (Long Short-Term Memory): A type of recurrent neural network designed to overcome the vanishing gradient problem and learn long-range dependencies in sequences.

Machine Translation (MT): The automatic translation of text or speech from one language to another using computational methods.

Neural Machine Translation (NMT): Machine translation systems that use neural networks, particularly deep learning models, to perform translation.

Parameter Efficient Fine-Tuning (PEFT): Techniques that enable fine-tuning of large models by training only a small subset of parameters, reducing computational requirements.

Pre-training: The initial training phase where a model learns general language patterns from large, unlabeled text corpora.

RNN (Recurrent Neural Network): A neural network architecture designed to process sequential data by maintaining hidden states that carry information from previous time steps.

Rule-Based Machine Translation (RBMT): Early machine translation systems that relied on handcrafted linguistic rules and dictionaries.

Self-Attention: A mechanism in transformer models where each position in a sequence can attend to all positions in the same sequence, enabling

parallel processing.

Sequence-to-Sequence (Seq2Seq): A neural network architecture designed to transform one sequence into another, commonly used in machine translation.

Statistical Machine Translation (SMT): Machine translation systems that learn translation patterns from bilingual text corpora using statistical methods.

Transformer: A neural network architecture introduced in 2017 that relies entirely on attention mechanisms, becoming the foundation for modern language models.

Transfer Learning: A machine learning technique where knowledge gained from one task is applied to improve performance on a related task.

Appendix: Historical BLEU Score Benchmarks

This appendix provides reference data on the progression of machine translation quality over time, as measured by BLEU scores on standard benchmarks.

```
- **Rule-Based MT (1990s)**: ~12 BLEU points
  - Early systems before BLEU metric introduction (2002)
  - Based on qualitative assessments and limited automated evaluation
- **Statistical MT (2000s)**: ~20 BLEU points
  - Early SMT systems on WMT benchmarks
  - Word-based alignment models
- **Phrase-Based SMT (2005-2010)**: ~25 BLEU points
  - Koehn et al. (2003) phrase-based models
  - Significant improvement over word-based approaches
- **Early Neural MT (2014-2015)**: ~26 BLEU points
```

```
  - Sutskever et al. (2014) Seq2seq with RNNs
  - First effective neural machine translation systems
- **Attention-Based NMT (2015–2016)**: ~28 BLEU points
  - Bahdanau et al. (2015) attention mechanism
  - Google NMT deployment (2016): 26.3 BLEU for English-German
- **Transformer Architecture (2017)**: ~29 BLEU points
  - Vaswani et al. (2017): 28.4 BLEU on WMT'14 English-German
  - Optimized training (2018): 29.3 BLEU
- **Modern LLMs (2020s)**: ~35 BLEU points
  - State-of-the-art systems achieve 35-45 BLEU for major language pairs
  - Approaching human-level performance (estimated 50-55 BLEU)
```

Sources:

- Papineni et al. (2002) — BLEU metric introduction
- Koehn et al. (2003) — Phrase-based translation
- Sutskever et al. (2014) — Seq2seq neural networks
- Bahdanau et al. (2015) — Attention mechanism
- Vaswani et al. (2017) — Transformer architecture
- WMT Shared Task Results (2006–2023)
- Google Research publications on NMT deployment

References

Academic Papers and Books

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. ICLR. https://arxiv.org/abs/1409.0473

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., … & Roossin, P. S. (1990). A statistical approach to machine

translation. https://aclanthology.org/J90-2002.pdf

Brown, P. F., Della Pietra, S. A., Della Pietra, V. J., & Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. Computational Linguistics, 19(2), 263–311. https://aclanthology.org/J93-2003.pdf

Castaño, M.-A., Casacuberta, F., & Vidal, E. (1997). Machine translation using neural networks and finite-state models. In Proceedings of the 7th Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages (TMIMTNL), St John's College, Santa Fe. Available at: https://aclanthology.org/1997.tmi-1.19/

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1724–1734). Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1179

Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. In Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (pp. 103–111). Association for Computational Linguistics. https://doi.org/10.3115/v1/W14-4012

Dai, A.M., & Le, Q.V. (2015). Semi-supervised Sequence Learning. ArXiv, abs/1511.01432. https://arxiv.org/abs/1511.01432

Devvrit, Kudugunta, S., Kusupati, A., Dettmers, T., Chen, K., Dhillon, I.S., Tsvetkov, Y., Hajishirzi, H., Kakade, S.M., Farhadi, A., & Jain, P. (2023). MatFormer: Nested Transformer for Elastic Inference. ArXiv, abs/2310.07707. https://arxiv.org/abs/2310.07707

Elman, J. L. (1990). Finding structure in time. Cognitive Science, 14(2), 179–211. https://doi.org/10.1207/s15516709cog1402_1

Gemma Team. (2025). Gemma 3N. Google DeepMind. https://ai.google.dev/gemma/docs/gemma-3n

Gemma Team. (2025). Gemma 3 Technical Report. ArXiv, abs/2503.19786. https://arxiv.org/abs/2503.19786

Google AI. (2024). Fine-tune Gemma using Hugging Face Transformers and QLoRA. Google AI for Developers. https://ai.google.dev/gemma/docs/core/huggingface_text_finetune_qlora

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hugging Face. (2023). Transformers: State-of-the-art machine learning for Pytorch, TensorFlow, and JAX. https://huggingface.co/transformers/

Hugging Face. (2024). Low-Rank Adaptation (LoRA). Hugging Face PEFT Documentation. https://huggingface.co/docs/peft/en/package_reference/lora

Hutchins, W. J. (2004, September 28-October 2). The Georgetown-IBM experiment demonstrated in January 1954. In R. E. Frederking & K. B. Taylor (Eds.), Proceedings of the 6th Conference of the Association for Machine

Translation in the Americas: Technical Papers (pp. 102–114). Springer. https://aclanthology.org/2004.amta-papers.12/

Hutchins, W. J. (2005). Machine translation: A concise history. Journal of Translation Studies, 8(1), 1–16.

Hutchins, J. (1997). From First Conception to First Demonstration: the Nascent Years of Machine Translation, 1947–1954. A Chronology. Machine Translation, 12(3), 195–252.

Koehn, P., Och, F. J., & Marcu, D. (2003). Statistical phrase-based translation. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (pp. 48–54). Association for Computational Linguistics. https://aclanthology.org/N03-1017/

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., … & Zens, R. (2007). Moses: Open source toolkit for statistical machine translation. In Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions (pp. 177–180). Association for Computational Linguistics. https://aclanthology.org/P07-2045/

Och, F. J. (2005). Statistical machine translation: From single-word models to alignment templates. PhD thesis, RWTH Aachen University.

Och, F. J., & Ney, H. (2004). The alignment template approach to statistical machine translation. Computational Linguistics, 30(4), 417–449. https://aclanthology.org/J04-4001/

Lyu, C., Xu, J., Wang, L., & Wu, M. (2023). A Paradigm Shift: The Future of Machine Translation Lies with Large Language Models. International Conference on Language Resources and Evaluation.

Locke, W., & Booth, A. D. (Eds.). (1955). Machine translation of languages: Fourteen essays. MIT Press. https://mitpress.mit.edu/9780262120029/machine-translation-of-languages/

Locke & Booth Issue the First Book on Machine Translation. (n.d.). History of Information. https://www.historyofinformation.com/detail.php?entryid=878

The Origin of Statistical Machine Translation: History of Information. (n.d.). https://www.historyofinformation.com/detail.php?id=681

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (pp. 311–318). Association for Computational Linguistics. https://aclanthology.org/P02-1040/

Radford, A., & Narasimhan, K. (2018). Improving Language Understanding by Generative Pre-Training. arXiv preprint arXiv:1802.05698.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21(140), 1–67. https://arxiv.org/abs/1910.10683

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536.

https://doi.org/10.1038/323533a0

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems 27 (NIPS 2014). https://arxiv.org/abs/1409.3215

Tiedemann, J. (2012). Parallel data, tools and interfaces in OPUS. In Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12) (pp. 2214–2218). European Language Resources Association. http://www.lrec-conf.org/proceedings/lrec2012/pdf/463_Paper.pdf

Tunstall, L., von Werra, L., & Wolf, T. (2022). Natural language processing with transformers (Revised ed.). O'Reilly Media.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), Advances in Neural Information Processing Systems 30 (pp. 5998–6008). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wang, H., Wu, H., He, Z., Huang, L., & Church, K. W. (2022). Progress in machine translation. Engineering, 18, 143–153. https://doi.org/10.1016/j.eng.2021.03.023

Weaver, W. (1955). Translation (1949). In W. N. Locke & A. D. Booth (Eds.), Machine translation of languages (pp. 15–23). MIT Press. https://www.historyofinformation.com/detail.php?entryid=878

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., & Raffel, C. (2020). mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer. North American Chapter of the Association for Computational Linguistics. https://arxiv.org/abs/2010.11934

Yu, Y., Yang, C.H., Kolehmainen, J., Shivakumar, P.G., Gu, Y., Ryu, S., Ren, R., Luo, Q., Gourav, A., Chen, I., Liu, Y., Dinh, T., Gandhe, A., Filimonov, D., Ghosh, S., Stolcke, A., Rastrow, A., & Bulyko, I. (2023). Low-Rank Adaptation of Large Language Model Rescoring for Parameter-Efficient Speech Recognition. 2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), 1–8. https://arxiv.org/abs/2309.15223

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2019). BERTScore: Evaluating text generation with BERT. arXiv preprint arXiv:1904.09675.

Datasets and Models

Helsinki-NLP/opus_books. https://huggingface.co/datasets/Helsinki-NLP/opus_books

mT5: A massively multilingual pre-trained text-to-text transformer (Xue et al., 2020). https://github.com/google-research/multilingual-t5, https://huggingface.co/google/mt5-base

Tutorials and Guides

PyTorch Sequence-to-Sequence Translation Tutorial. https://docs.pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

PyTorch TorchText Translation Tutorial. https://docs.pytorch.org/tutorials/beginner/torchtext_translation.html?highlight=transformer

PyTorch Sequence-to-Sequence Implementation. https://github.com/bentrevett/pytorch-seq2seq/blob/main/3%20-%20Neural%20Machine%20Translation%20by%20Jointly%20Learning%20to%20Align%20and%20Translate.ipynb

Originally published at https://www.ranton.org.

AI Translation NLP Greek Machine Translation

Written by Richard Anton

130 followers · 800 following

Tech Fellow at Walmart. Previously at Snowflake on Streamlit and Notebooks. Former Amazonian and Google SRE. Interested in NLP, ML/AI, & gfx.

Follow

Responses (1)

Alex Mylnikov

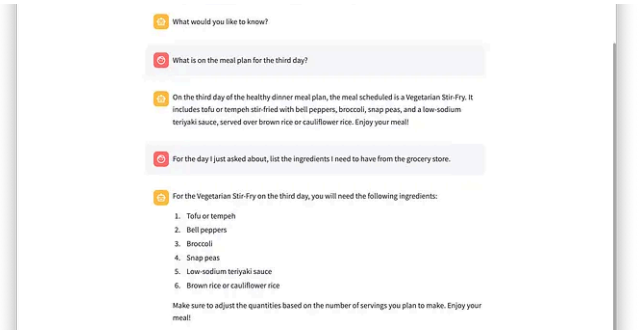
What are your thoughts?

Ethan Miller he
2 days ago

Comprehensive overview of machine translation! The balance between theory and practical applications is well executed.

Reply

More from Richard Anton

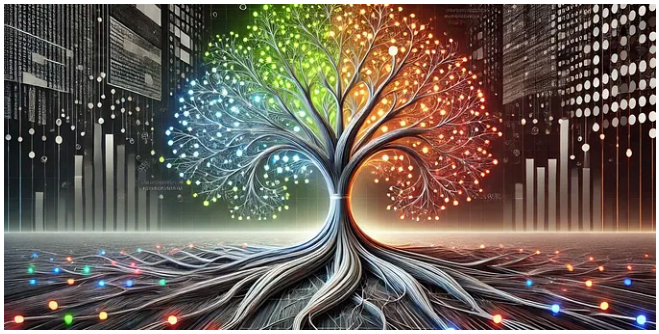


In Snowflake Builders Blog: Data Engin... by Richa...

LangChain and Streamlit RAG

In this blog post, we will explore how to use Streamlit and LangChain to create a chatbot...

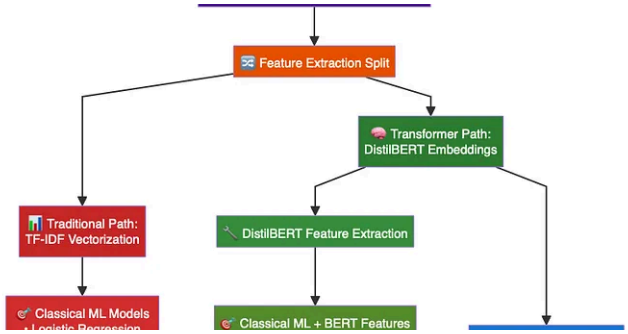
Apr 22, 2024 175 2



Richard Anton

Gradient-Boosted Machines (GBMs): Fundamentals and...

Dec 28, 2024 6



Richard Anton

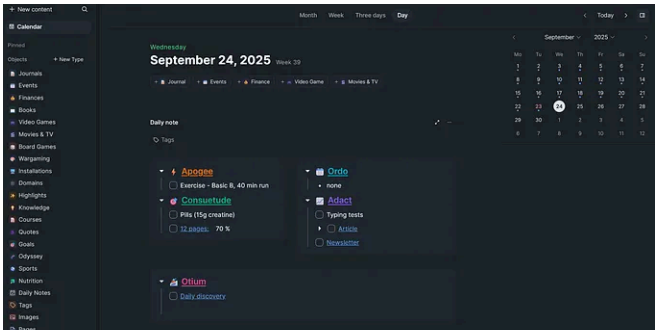
Classifying Concord

Jun 16 151 3

See all from Richard Anton

Recommended from Medium

Bogdan Ilyin



Tosny

7 Websites I Visit Every Day in 2025

Denmark Just Triggered Putin's Worst Nightmare

Europe's quietest country just made one of the loudest moves against Moscow's war...

Oct 6 14.2K 177

If there is one thing I am addicted to, besides coffee, it is the internet.

Sep 23 5.3K 191

Scott Galloway

How Does the End Begin?

The top 10 stocks in the S&P 500 account for 40% of the index's market cap. Since ChatGP...

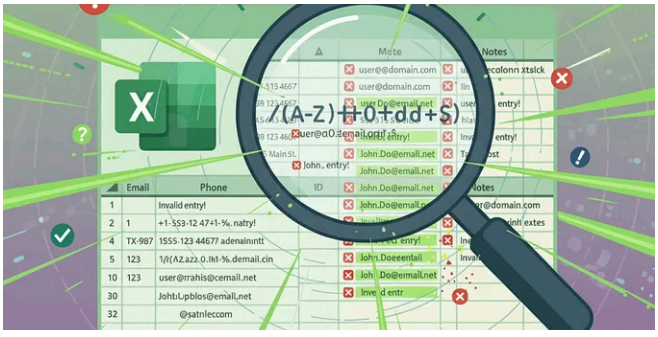
4d ago 2.2K 40

aakash

Perplexity Just Unleashed 10 FREE AI Agents That Do Your Entire Job...

Stop what you are doing. The era of the simple AI search engine is officially over.

Oct 7 773 29



In Intuition by James Wilkins

How Far Can You Push Excel with Regex?

If you've spent any time in Excel, you'll know the struggle of working with a messy dataset...

In Fourth Wave by Maria Cassano

I Went to a No Kings Protest at a Church, and What I Saw Shocked...

For context, I'm a liberal ex-Christian feminist in a Republican town