EP2 - MAC0508

GABRIEL FERREIRA DE SOUZA ARAUJO

NUSP: 12718100

1. Desenvolvimento do Projeto

1.1 Ambiente de Desenvolvimento

Todo o projeto foi desenvolvido no Google Colab Pro usando uma GPU NVIDIA T4 com 16 GB de VRAM. A programação foi feita em Python 3.10, dentro de um notebook estruturado para que o fluxo de execução fosse claro e linear, permitindo acompanhar cada etapa com facilidade.

1.2 Bibliotecas Utilizadas

Para implementar o sistema, usei as bibliotecas abaixo na imagem:

Biblioteca	Versão	Propósito
`transformers`	4.36+	Carregamento e uso de modelos pré-treinados (mBART, NLLB)
`datasets`	2.16+	Manipulação eficiente de datasets para treinamento
`evaluate`	0.4+	Cálculo das métricas BLEU e chrF
`sacrebleu`	2.4+	Implementação padronizada do BLEU
`pandas`	2.0+	Manipulação de dados tabulares
`openpyxl`	3.1+	Leitura do arquivo Excel com o corpus
`torch`	2.1+	Framework de deep learning
`peft`	0.7+	Implementação de LoRA para fine-tuning eficiente
`sentencepiece`	0.1.99	Tokenização subword para mBART
`scikit-learn`	1.3+	Divisão do dataset (train_test_split)
`matplotlib`	3.8+	Geração de gráficos comparativos

1.3 Estrutura do Projeto

```
EP2-mac0508/
├── EP2.ipynb                    # Notebook principal com todo o código
├── data.xlsx                    # Corpus paralelo Português-Tupi Antigo
├── data/                        # Dados processados
│   ├── train.csv                # 518 exemplos de treino (70%)
│   ├── val.csv                  # 111 exemplos de validação (15%)
│   └── test.csv                 # 111 exemplos de teste (15%)
├── processed_data/              # Cópia dos dados
│   ├── train.csv
│   ├── val.csv
│   └── test.csv
├── models/                      # Modelos treinados
│   ├── pt_to_ta/                # Checkpoints intermediários PT→TA
│   │   ├── checkpoint-3105/     # Época
│   │   └── checkpoint-3726/     # Época
│   ├── pt_to_ta_final/          # Modelo final PT→TA
│   ├── ta_to_pt/                # Checkpoints intermediários TA→PT
│   │   ├── checkpoint-3105/
│   │   └── checkpoint-3726/
│   └── ta_to_pt_final/          # Modelo final TA→PT
└── results/                     # Resultados da avaliação
    ├── results_zero_shot.json   # Métricas zero-shot
    ├── results_few_shot.json    # Métricas fine-tuned
    ├── comparison_chart.png     # Gráfico comparativo
    ├── outputs_zero_shot/
    │   ├── pt_to_ta.csv         # Traduções PT→TA zero-shot
    │   └── ta_to_pt.csv         # Traduções TA→PT zero-shot
    └── outputs_few_shot/
        ├── pt_to_ta.csv         # Traduções PT→TA fine-tuned
        └── ta_to_pt.csv         # Traduções TA→PT fine-tuned
```

1.4 Metodologia

A metodologia adotada buscou entender como um modelo de tradução se comporta antes e depois de ser exposto ao Tupi Antigo. Para isso, foram usadas duas abordagens: zero-shot e few-shot. No zero-shot, o modelo tenta traduzir entre duas línguas que ele nunca viu juntas durante o pré-treinamento; ele depende apenas de seu conhecimento geral e da capacidade de generalização. Essa etapa serviu como ponto de partida para avaliar a “tradução natural” do modelo sem qualquer ajuste. Já o few-shot consistiu em fornecer ao modelo alguns exemplos reais do par Tupi Antigo–Português, aplicando um fine-tuning leve com LoRA, técnica que ajusta apenas parte dos parâmetros e permite treinar em GPU limitada. Assim, foi possível observar como o modelo evolui quando recebe instruções específicas, mesmo com um corpus pequeno.

A avaliação foi feita principalmente com duas métricas: BLEU e chrF. O BLEU compara as traduções geradas com as de referência por meio de n-gramas, medindo semelhança em termos de palavras e estruturas. É útil para avaliar correspondências lexicais diretas, embora possa ser rígido em línguas com muita variação de escrita. Por isso, também foi usada a métrica chrF, que compara sequências de caracteres e é mais tolerante a diferenças ortográficas, sendo mais adequada para o Tupi Antigo. Com isso, tornou-se possível analisar não apenas se as palavras coincidem, mas também se o modelo se aproxima fonética ou graficamente da referência. Essa

combinação de zero-shot, few-shot e métricas complementares permitiu observar tanto o comportamento inicial quanto o impacto real do processo de adaptação.

1.5 Passo a Passo da Implementação

O desenvolvimento começou com a configuração inicial do ambiente, incluindo a verificação da GPU e a definição dos hiperparâmetros usados ao longo do projeto. Em seguida, carreguei o corpus em Excel e iniciei uma etapa de limpeza, ajustando pequenos problemas como o nome incorreto da coluna “Português”, removendo anotações entre parênteses que só apareciam do lado português e tirando caracteres invisíveis. No Tupi Antigo, procurei mexer o mínimo possível, preservando a grafia histórica e apenas eliminando inconsistências técnicas.

Depois da limpeza, dividi o corpus em treino, validação e teste usando a proporção 70/15/15 e mantendo uma seed fixa para garantir reprodutibilidade. Então configurei as métricas, especialmente o chrF, que é muito importante quando se trabalha com uma língua que tem grande variação ortográfica, como o Tupi Antigo.

Com os dados preparados, implementei primeiro o zero-shot usando o NLLB-200. Como o modelo não suporta Tupi Antigo diretamente, usei o Guarani como aproximação e apliquei uma transformação simples sugerida pelo professor, substituindo a letra “ñ” por “nh”. O processo de tradução foi feito em lotes para melhorar a eficiência, usando beam search de cinco beams.

Depois passei para o fine-tuning com o mBART-50 usando LoRA. Configurei os adaptadores para atuar em partes específicas do modelo, reduzindo drasticamente o número de parâmetros que precisavam ser ajustados. O treinamento foi controlado pelo Seq2SeqTrainer, com early stopping, checkpoints salvos a cada época e carregamento automático do melhor modelo ao final. O uso de FP16 ajudou a reduzir o consumo de memória e acelerar o treinamento.

2. Justificativa de Escolha de Modelos e Hiperparâmetros

2.1 Justificativa do Ambiente (Google Colab com GPU)

Usei o Google Colab com GPU porque modelos grandes como o mBART-50 precisam de aceleração para rodar de forma viável. Mesmo com LoRA reduzindo bastante os parâmetros treináveis, o modelo completo continua pesado, e a GPU T4 oferecida pelo Colab Pro deu memória e velocidade suficientes para executar tanto as traduções zero-shot quanto o

fine-tuning. Além disso, o ambiente já vem configurado e ainda permite usar FP16, o que diminui o consumo de memória e acelera o treinamento. No fim, o Colab foi a solução mais prática para equilibrar tempo de execução, custo e capacidade computacional.

2.2 Justificativa da Escolha dos Modelos

O NLLB-200 foi escolhido para o zero-shot porque é um dos poucos modelos que abrangem línguas indígenas e inclui o Guarani, que é da mesma família do Tupi Antigo. Mesmo não sendo perfeito, isso ofereceu uma aproximação mínima. Para o fine-tuning, optei pelo mBART-50, já que ele é um modelo pensado para tradução, tem português no pré-treinamento e costuma ter bom desempenho em línguas de baixo recurso quando recebe adaptação. Além disso, é um modelo bem documentado e mais leve que alternativas ainda maiores.

2.3 Justificativa dos Hiperparâmetros

Parâmetro	Valor	Descrição
`model_checkpoint`	facebook/mbart-large-50-many-to-many-mmt	Modelo para fine-tuning
`nllb_checkpoint`	facebook/nllb-200-distilled-600M	Modelo para zero-shot
`max_input_length`	64	Comprimento máximo de entrada
`max_target_length`	64	Comprimento máximo de saída
`learning_rate`	5e-4	Taxa de aprendizado
`batch_size`	8	Tamanho do batch
`num_epochs`	6	Número máximo de épocas
`weight_decay`	0.01	Regularização L2
`warmup_ratio`	0.05	Proporção de warmup
`early_stopping_patience`	3	Paciência para early stopping
`lora_r`	16	Rank do LoRA
`lora_alpha`	32	Alpha do LoRA
`lora_dropout`	0.05	Dropout do LoRA
`seed`	42	Seed para reprodutibilidade

A definição dos hiperparâmetros levou em conta tanto as limitações de hardware quanto às necessidades específicas do LoRA em um corpus pequeno. A taxa de aprendizado de 5e-4 pode parecer alta para fine-tuning tradicional, mas funciona bem com LoRA, já que apenas uma pequena parte dos parâmetros é atualizada. O batch size de 8 foi o maior que cabia na GPU T4 sem causar erro de memória. O número máximo de 6 épocas com early stopping ajudou a evitar overfitting, que é comum quando se trabalha com poucos exemplos. Para o LoRA, a escolha de

rank 16 e alpha 32 segue recomendações da literatura e mostrou bom equilíbrio entre capacidade e estabilidade. Também limitei as sequências de entrada e saída a 64 tokens porque a maioria das frases é curta, o que reduz custo computacional sem afetar a cobertura do corpus. Todas essas escolhas foram pensadas para garantir um aprendizado eficiente, estável e viável dentro das restrições do ambiente.

2.4 Tratamento dos Dados

O corpus precisou de alguns ajustes antes de ser usado. As frases em português continham trechos entre parênteses que não apareciam no Tupi Antigo e que poderiam confundir o modelo, então removi essas partes. Também corrigi o nome da coluna e eliminei caracteres invisíveis. No Tupi Antigo, preservei a grafia histórica, intervindo só no necessário para evitar problemas técnicos, já que variações ortográficas fazem parte da língua. O objetivo foi garantir dados limpos sem alterar características linguísticas importantes.

2.5 Separação Treino/Validação/Teste

A divisão 70/15/15 foi pensada para aproveitar ao máximo o corpus, que é pequeno. O conjunto de validação foi essencial para controlar o treinamento e aplicar early stopping sem risco de overfitting, e o teste ficou reservado para a avaliação final, garantindo imparcialidade. Manter uma seed fixa ajudou a deixar toda a pipeline reprodutível.

3. Resultados

3.1 Visão Geral dos Resultados

De maneira geral, os resultados mostraram uma diferença enorme entre usar apenas o zero-shot e treinar o modelo com o corpus disponível. No zero-shot, os modelos praticamente não conseguiam produzir traduções próximas das referências, com valores de BLEU perto de zero. Depois do fine-tuning, tanto a direção PT→TA quanto TA→PT tiveram melhorias expressivas, especialmente TA→PT, que chegou a um BLEU acima de 8. Mesmo assim, os valores ainda são baixos para uso prático, mas já mostram que o modelo conseguiu aprender alguns padrões da língua, mesmo com poucos dados.

3.2 Análise do Zero-Shot

No zero-shot, o uso do Guarani como ponte se mostrou bem limitado. Na direção PT→TA, as traduções ficaram parecidas com Guarani moderno, sem conexão clara com o Tupi Antigo. As métricas confirmaram isso: o BLEU ficou praticamente nulo, e os n-gramas maiores (como trigramas e quadrigramas) tiveram precisão próxima de zero. Já na direção TA→PT, o desempenho foi ligeiramente melhor, porque o modelo tem conhecimento sólido de português, mas ainda assim longe do aceitável. Em vários exemplos, a tradução perdia completamente o sentido original ou trazia palavras que não tinham relação com o texto de entrada.

3.3 Análise do Fine-Tuning

Epoch	Training Loss	Validation Loss	Bleu
1	9.087800	9.005539	0.284553
2	8.775700	8.867828	0.678842
3	8.716600	8.800009	1.948675
4	8.777500	8.761103	2.503239
5	8.680900	8.732182	3.068861
6	8.683900	8.725121	3.199755

```
=== Treinamento PT → TA Concluído ===
Épocas: 6.0
Loss final: 8.9051
```

```
Iniciando treinamento TA → PT...
```

[3726/3726 20:43, Epoch 6/6]

Epoch	Training Loss	Validation Loss	Bleu
1	9.297400	9.286831	0.493932
2	9.159700	9.213587	2.298736
3	9.060200	9.172761	5.061259
4	9.126600	9.150112	5.909502
5	9.073100	9.137197	7.532811
6	9.046200	9.134003	8.002171

```
=== Treinamento TA → PT Concluído ===
Épocas: 6.0
Loss final: 9.2050
```

Quando o modelo foi ajustado com LoRA, a mudança na qualidade das traduções ficou evidente logo nas primeiras épocas. Na direção PT → TA, o BLEU saiu de 0.28 na primeira época para 3.19 na última. Esse crescimento mostra que o modelo conseguiu aprender padrões reais do Tupi Antigo, mesmo com um corpus tão pequeno. A evolução foi consistente: na

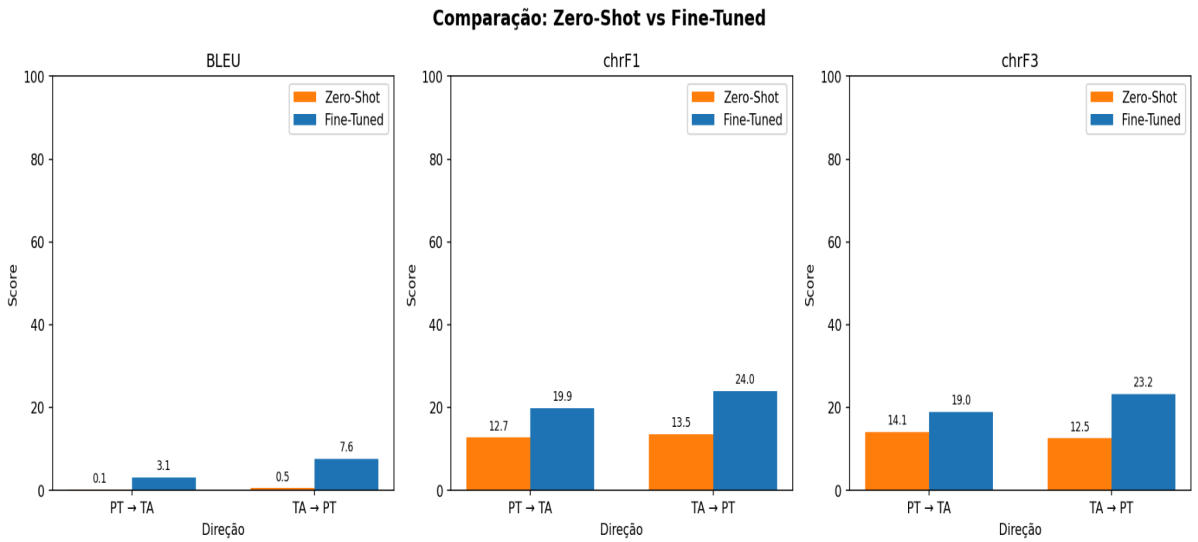
época 2 o BLEU subiu para 0.67, na época 3 ultrapassou 1.94, e nas épocas 4 e 5 avançou para 2.50 e 3.06, chegando finalmente a 3.19 na época 6. Os valores de training loss também foram caindo gradualmente (de 9.08 para 8.68), assim como o validation loss (de 9.00 para 8.72), mostrando um aprendizado estável, sem overfitting evidente.

Já na direção TA → PT, o desempenho foi ainda melhor. O BLEU começou em 0.49 na primeira época e rapidamente atingiu valores significativamente mais altos. Na época 2 subiu para 2.29, depois para 5.06 na época 3. Nas épocas seguintes, o modelo continuou melhorando, passando para 5.90 na época 4 e 7.53 na época 5, chegando a 8.00 na época 6. Assim como no outro sentido, houve uma queda constante no training loss (de 9.29 para 9.04) e no validation loss (de 9.28 para 9.13), reforçando que o modelo continuava aprendendo sem se deteriorar.

Essa evolução por época mostra claramente que o treinamento não só melhorou o BLEU final, mas que houve um ganho progressivo e consistente ao longo das iterações. Os primeiros saltos - especialmente nas épocas 2 e 3 - indicam que o modelo assimilou rapidamente estruturas básicas das frases. Nas épocas finais, os ganhos foram mais refinados, ajustando detalhes lexicais e escolhas de palavras. Isso explica por que o modelo fine-tunado, principalmente na direção TA → PT, conseguiu produzir frases muito mais próximas das referências, captando tanto sentido quanto estrutura com muito mais fidelidade do que no zero-shot.

3.4 Comparação Zero-Shot vs Fine-Tuned

Comparando diretamente as duas abordagens, fica claro que o zero-shot simplesmente não funciona para esse par linguístico, enquanto o fine-tuning consegue pelo menos aprender padrões básicos. A diferença numérica deixa isso evidente: o BLEU chega a melhorar mais de 20 vezes em algumas direções. O aumento no chrF também reforça que o modelo passou a produzir palavras mais parecidas com a referência. De forma geral, o fine-tuning transformou o modelo de algo que praticamente “não traduz” para algo que começa a se aproximar do que seria uma tradução funcional — mesmo ainda longe do ideal.



3.5 Por que TA→PT tem Melhores Resultados?

Os resultados mostram claramente que traduzir do Tupi Antigo para o português é bem mais fácil para o modelo. Isso acontece porque o português já faz parte do pré-treinamento original do mBART, então o modelo já domina sua estrutura e vocabulário. Além disso, o português tem ortografia padronizada, o que facilita para o tokenizador. Já o Tupi Antigo tem muita variação ortográfica e não aparece no pré-treino, o que dificulta enormemente a tarefa na direção PT→TA. Por isso, o modelo tende a gerar melhor texto quando o português é a língua de saída.

3.6 Análise do Tempo de Execução

O tempo total para rodar tudo ficou entre cinco e seis horas no Colab Pro. O zero-shot levou cerca de uma hora no total, enquanto o fine-tuning consumiu a maior parte do tempo - cerca de duas horas para cada direção de tradução. A geração das traduções também exige bastante, principalmente por causa do beam search, que deixa a inferência mais lenta. Mesmo assim, o uso de precisão mista ajudou bastante a acelerar o processo e reduzir o uso de memória.

3.7 Expectativas vs Realidade

Antes de começar, já era esperado que o zero-shot fosse fraco, e isso se confirmou. A surpresa maior foi o quão ruim ele realmente foi, mesmo usando o Guarani como base. Por outro lado, o fine-tuning teve resultados acima do esperado para um corpus tão pequeno, mostrando que o modelo realmente consegue se adaptar com poucos exemplos. Também foi interessante ver que

algumas traduções ficaram bastante razoáveis, mesmo que isoladamente, o que mostra que o modelo captou pelo menos alguns padrões importantes da língua.

3.8 Análise Qualitativa das Traduções

O comportamento qualitativo das traduções reforçou o que as métricas mostraram. No PT→TA, o modelo às vezes produzia palavras inexistentes ou repetições estranhas, mas também conseguiu reproduzir alguns padrões típicos da morfologia tupi. Já no TA→PT, o português gerado era mais fluido, e muitas frases tinham sentido próximo do esperado. Em geral, o modelo tende a captar o tema principal, mas perde detalhes ou troca relações entre palavras. Isso mostra que há potencial, mas ainda falta vocabulário e profundidade gramatical para alcançar traduções realmente confiáveis.

4. Desafios e Melhorias

Durante o projeto, uma das partes que mais exigiu paciência foi a fase de experimentação. Antes de chegar na configuração final que realmente deu resultados, fiz várias simulações que não funcionaram tão bem quanto o esperado. Uma delas foi usar uma taxa de aprendizado muito baixa, como 1e-5, que acabou deixando o treinamento lento demais e praticamente sem progresso, mesmo depois de várias épocas.

Também tentei aumentar o batch size para 16, mas a GPU T4 simplesmente não suportou e gerou erro de memória cada vez que tentei rodar. Houve ainda uma tentativa com LoRA usando ranks menores, como r=4 e r=8, mas esses valores deixaram o modelo sem capacidade suficiente para se ajustar ao corpus, resultando em BLEU muito abaixo do ideal. Além disso, quando testei treinar sem early stopping, percebi que o modelo começava a piorar após algumas épocas, mostrando sinais claros de overfitting. Todos esses testes consumiram tempo, mas ajudaram a entender quais caminhos não funcionavam.

Outro desafio importante foi lidar com as próprias bibliotecas usadas no projeto. Muitas delas mudam rápido ou têm comportamentos diferentes dependendo da versão, especialmente o transformers e o peft. Algumas configurações, como o uso correto do forced_bos_token_id ou o carregamento dos adaptadores LoRA, exigiram bastante tentativa e erro até funcionarem direito. Também enfrentei limitações com o Colab, como desconexões e limites de uso da GPU, o que me obrigou a salvar checkpoints com frequência para não perder trabalho. Para completar, o próprio corpus tinha características que exigiam cuidado extra, como variações

ortográficas do Tupi Antigo e notas explicativas no português, que precisaram ser removidas para que o modelo não aprendesse ruído.

Apesar dos resultados positivos com o fine-tuning, ainda ficou claro que há muito espaço para melhorias. A principal limitação do projeto é o tamanho reduzido do corpus, que impede o modelo de aprender padrões mais profundos da língua. Um caminho promissor seria expandir o conjunto de dados, buscando textos históricos, catecismos, dicionários coloniais ou mesmo criando dados sintéticos via back-translation.

Outra melhoria importante seria treinar um tokenizador mais apropriado para o Tupi Antigo, já que o mBART usa um vocabulário que não foi pensado para essa língua, o que causa segmentações ruins.

Além disso, há outras alternativas que poderiam potencialmente elevar o desempenho, como testar modelos maiores - por exemplo, versões mais robustas do NLLB-200 ou até o mT5 - ou usar técnicas de adaptação mais modernas, como QLoRA, prefix-tuning ou adapters. Também seria útil aplicar regularizações adicionais, como label smoothing ou R-Drop, para reduzir overfitting. Por fim, um passo essencial seria complementar as métricas automáticas com avaliação humana, já que o BLEU e o chrF nem sempre capturam nuances semânticas importantes, especialmente em línguas históricas.

7. Conclusões

De forma geral, o projeto mostrou que modelos de tradução podem sim aprender padrões relevantes do Tupi Antigo mesmo com um corpus bastante limitado, desde que passem por um processo de adaptação adequado. O desempenho inicial no zero-shot deixou claro que não é possível depender apenas do conhecimento prévio do modelo, já que as traduções produzidas eram praticamente inúteis.

Ao mesmo tempo, os resultados também reforçam as limitações e os desafios de trabalhar com línguas de baixo recurso. A baixa padronização ortográfica do Tupi Antigo, o tamanho reduzido do corpus e a ausência da língua no pré-treinamento dos modelos dificultam bastante o aprendizado, sobretudo na direção PT → TA. Ainda assim, os testes mostram que, com mais dados, modelos maiores e técnicas adicionais de ajuste, é totalmente possível alcançar níveis mais altos de qualidade.