Guilherme Busato Vecchi, Thábata Pontes Lazarou Risso

Solução do Desafio de Esquemas de Winograd em Português Brasileiro

São Paulo

2020

Guilherme Busato Vecchi, Thábata Pontes Lazarou Risso

Solução do Desafio de Esquemas de Winograd em Português Brasileiro

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

São Paulo
2020

Guilherme Busato Vecchi, Thábata Pontes Lazarou Risso

Solução do Desafio de Esquemas de Winograd em Português Brasileiro

Trabalho de Conclusão de Curso apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

Área de Concentração: Engenharia de Computação

Orientador: Glauber de Bona

São Paulo

2020

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

Catalogação-na-publicação

Vecchi, Guilherme Busato
Solução do Desafio de Esquemas de Winograd em Português Brasileiro / G. B. Vecchi, T. P. L. Risso -- São Paulo, 2020.
94 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Desafio de Esquemas de Winograd 2.Inteligência Artificial 3.Processamento de Linguagem Natural I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Risso, Thábata Pontes Lazarou

RESUMO

O desafio de esquemas de Winograd é composto por uma série de esquemas com uma sentença ambígua cuja ambiguidade é facilmente solucionável por um falante nativo da língua, por possuir senso comum, mas o mesmo não pode ser dito para uma máquina. Desta forma, este desafio visa testar a capacidade de obtenção do senso comum em algoritmos de processamento de linguagem natural. Com isso, o objetivo deste trabalho é avançar o estado da arte existente em português brasileiro, que possui uma acurácia de aproximadamente 45% nos 273 esquemas que compõe o desafio, visto que soluções na língua inglesa, por exemplo, já chegam em aproximadamente 90% de acurácia, principalmente devido a grande evolução da língua em tarefas de processamento de linguagem natural. Este objetivo é alcançado a partir de duas abordagens, a primeira focada em entender a possibilidade de uso do desenvolvimento em língua inglesa para a solução do desafio em português, e a segunda focada em uma solução usando modelos de linguagem em português, obtendo 72,9% e 65,9% de acurácia, respectivamente.

Palavras-chave: Desafio de Esquemas de Winograd. Inteligência Artificial. Processamento de Linguagem Natural.

ABSTRACT

The Winograd Schema Challenge consists of a series of schemas with an ambiguous sentence whose ambiguity is easily solved by a native speaker of the language, for having common sense, but the same cannot be said for a machine. Thus, the challenge aims to test natural language processing algorithms' capacity of acquiring common sense. Thereby, the goal of this project is to advance the current state of the art in Brazilian Portuguese, whose accuracy is approximately 45,5% on the 273 schemas that compose the challenge, given that solutions in the English language, for example, already reach approximately 90% accuracy, mainly due to the great development of the language on natural language processing tasks. This goal is achieved using two approaches, the first one focused on understanding the possibility of using the English language development for solving the challenge in Portuguese, and the second focused on using Portuguese language models for the solution, achieving accuracies of 72,9% and 65,9%, respectively.

Keywords: Winograd Schema Challenge. Artificial Intelligence. Natural Language Processing.

LISTA DE TABELAS

Tabela 6.1 - Resultados do BLEU para tradução dos esquemas originais, em português…….....42
Tabela 6.2 - Porcentagem de sentenças idênticas ao dataset original…..…..…...43
Tabela 6.3 - Comparativo de resultados entre modelos e datasets utilizados……..54
Tabela 6.4 - Acurácia para os casos em que a máscara foi substituída corretamente……...64
Tabela 6.5 - Precisão de cada caso de tradução, descontando-se casos com [MASK] errado…….64
Tabela 6.6 - Comparação entre as acurácias obtidas e os tempos de execução para o treinamento de ajuste fino por uma época, para diferentes tamanho de lote (batch-size).....77
Tabela 6.7 - Comparação entre as acurácias obtidas para o treinamento de ajuste fino batch-size 16, para 7 épocas de treinamento..……...80
Tabela 7.1 - Resultados obtidos a partir da Abordagem 1……...85
Tabela 7.2 - Resultados obtidos a partir da Abordagem 2……...85

LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BLEU	Bilingual Evaluation Understudy
CPU	Central Processing Unit
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
JSON	JavaScript Object Notation
JSONL	JSON Lines
MLM	Masked Language Modeling
NLTK	Python Natural Language Toolkit library
NSP	Next Sentence Prediction
PLN	Processamento de Linguagem Natural
WSC	Winograd Schema Challenge

SUMÁRIO

Introdução **11**

1.1. Objetivo 11

1.2. Motivação 12

1.3. Justificativa 13

1.4. Organização do Trabalho 14

Aspectos Conceituais **15**

2.1. Desafio de Esquemas de Winograd 15

2.2. Desafios de Esquemas de Winograd em Português Brasileiro 17

2.3. BLEU: Bilingual Evaluation Understudy 18

2.4 Processamento de Linguagem Natural 21

2.4.1. Modelos de Linguagem 22

2.4.2. Transfer Learning 23

2.5. BERT: Bidirectional Encoder Representations from Transformers 24

2.5.1. Arquitetura do BERT 26

2.5.2. Pré-treinamento do modelo 28

2.5.3. Casos de Uso do BERT 29

Metodologia do Trabalho **31**

Tecnologias Utilizadas **33**

Especificação **35**

5.1. BLEU 35

5.2. Especificação no Desafio de Esquemas de Winograd 36

Implementação **38**

6.1. Abordagem 1: Tradução dos Esquemas de Português para Inglês e Execução do Modelo de Linguagem em Inglês 38

6.1.1 Tradução dos esquemas para o inglês 38

6.1.2 Metrificando a tradução 40

6.1.3. Ferramentas 43

6.1.4. Implementação 45

6.1.4.1. Preparação do Ambiente no Colab 45

6.1.4.2. Entendimento da Solução em Código Aberto 48

6.1.5. Avaliação dos resultados com tradução otimizada 54

6.2. Abordagem 2: Execução do Modelo de Linguagem em Português com ajuste fino no WINOGRANDE traduzido 66

6.2.1 Tradução do WINOGRANDE para o português 66

6.2.2. Implementação 72

6.2.3. BERT Multilingual 75

6.2.3.1. Execução do BERT Multilingual sem ajuste fino 76

6.2.3.2. Execução do BERT Multilingual com ajuste fino 76
6.2.4. BERT Portuguese 82
6.2.4.1. Execução do BERT Portuguese sem ajuste fino 83
6.2.4.2. Execução do BERT Portuguese com ajuste fino 83
Análise de Resultados 85
Conclusão 89
Referências 91
APÊNDICE A - Documentação do portal para demonstração prática 92

1. Introdução

O desafio de esquemas de Winograd, proposto em [1], cuja solução é o foco do presente trabalho, é composto por sentenças que possuem dois substantivos de mesma característica (gênero e número) e um pronome pessoal ou possessivo que se refere a um desses substantivos mas que, gramaticalmente, pode se referir ao outro. Isso causa uma ambiguidade que é facilmente solucionável por um ser humano falante do idioma, por possuir senso comum, enquanto que, para uma máquina, não se pode dizer o mesmo. Um exemplo pode ser visto abaixo:

O advogado fez uma pergunta ao acusado, mas ele estava relutante em respondê-la.

Quem estava relutante em respondê-la?

a. advogado

b. acusado

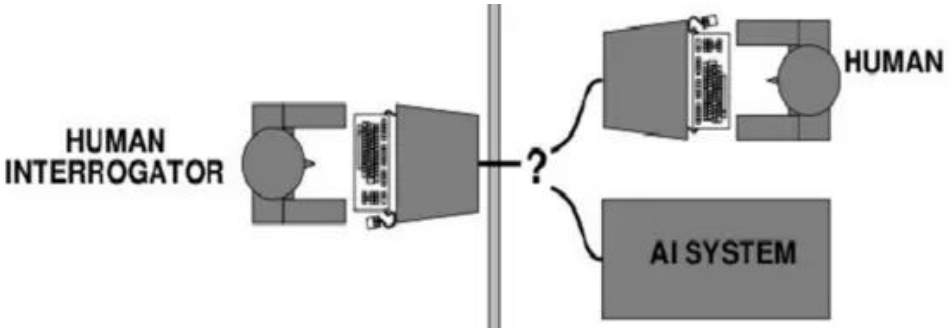
1.1. Objetivo

Este trabalho de formatura tem como objetivo desenvolver uma solução satisfatória para o desafio de esquemas de Winograd traduzido para português brasileiro, disponível em [2]. Tendo isso em vista, o objetivo é encontrar uma solução que se torne o estado da arte em português brasileiro, ou seja, que represente a referência dentre os modelos desenvolvidos para o desafio, a partir de melhores métricas de desempenho, aproximando-se do resultado de um ser humano no desafio de esquemas de Winograd. Além disso, um segundo objetivo é auxiliar no desenvolvimento do processamento de linguagem natural em português brasileiro, investigando a possibilidade de transferência de conhecimento entre línguas para a solução do desafio, dado que as soluções em inglês possuem um desempenho mais satisfatório quando comparadas às soluções em português brasileiro.

1.2. Motivação

Um dos grandes estímulos ao desenvolvimento da inteligência artificial é emular os pensamentos e atitudes de um ser humano. Diversos desafios foram propostos para colocar esse comportamento dos algoritmos a prova, sendo um dos mais famosos o teste que ficou popularmente conhecido como Teste de Turing, proposto em [3], apelidado com o nome de seu autor. Neste teste, em seu exemplo ilustrativo original, como pode ser visto na figura 1.1, um ser humano, que possui a função de juiz (*"Human Interrogator")*, se comunica via texto em linguagem natural com um outro humano (*"Human"*) e uma máquina, projetada para produzir respostas como se fosse um ser humano (*"AI System"*).

Figura 1.1: Esquema da aplicação do teste de Turing



Fonte: IA Expert Academy. Disponível em <https://iaexpert.academy/2016/07/19/historico-da-ia-teste-de-turing/>. Acesso em: 03 dez. 2020.

Diz-se que um algoritmo passou no Teste de Turing se o juiz não for capaz de distinguir entre o humano e a máquina. Segundo o teste, neste caso, a máquina consegue pensar por si própria. Vale ressaltar que este teste não analisa se uma máquina dá respostas corretas às perguntas do juiz, mas se a máquina tem um comportamento de resposta semelhante ao de um ser humano.

Este teste, porém, possui alguns aspectos que podem ser criticados na busca do entendimento da capacidade de pensamento das máquinas. Um dos pontos principais é o de que a máquina, para "enganar" e convencer o juiz, deve adotar uma identidade falsa, podendo criar diversas respostas para perguntas pessoais (como, por exemplo, nome, gênero, idade etc). Isto é natural para este

teste, mas, idealmente, gostaria de se saber se a máquina está “pensando” sem uma identidade falsa. Além disso, um teste no formato de conversa pode facilitar enganos e trapaças. Por fim, usar um juiz humano pode inserir questões interpretativas e subjetivas, o que dificulta a padronização do teste.

Neste contexto, surge o Desafio de Esquemas de Winograd, proposto em [1], um teste baseado em um questionário em formato de texto (originalmente produzido em inglês), com uma grande variedade de frases com ambiguidades gramaticais que são solucionáveis por senso comum, mas praticamente impossíveis de serem solucionáveis apenas gramaticalmente, no qual falantes nativos passam facilmente e que é administrado e classificado sem juízes humanos. Assim como no Teste de Turing, quando se é aprovado no teste, conclui-se que aquele que o realizou está, de certa forma, pensando (seja ser humano ou máquina).

1.3. Justificativa

O Processamento de Linguagem Natural (PLN) vem sendo amplamente utilizado em várias aplicações como análise de sentimento, *chatbots*, reconhecimento de voz, dentre outras várias que são objeto de estudo da Inteligência Artificial.

Desafios de inteligência artificial são muito importantes para incentivar o desenvolvimento de algoritmos em tarefas específicas, de modo a generalizar os benefícios obtidos para uso em outras aplicações. O Desafio de Esquemas de Winograd é um desafio focado na área de processamento de linguagem natural, composto por um conjunto de frases ambíguas que, para terem a ambiguidade resolvida corretamente, necessitam de conhecimento do senso-comum, intrínseco à natureza humana. Desta forma, o desenvolvimento neste desafio auxilia na inserção deste tipo de conhecimento nos algoritmos de PLN, o que é importante para que estes algoritmos possam compreender e emular, de forma mais adequada, a comunicação humana.

Entretanto, apesar de não ser um novo conceito e já ser usado no Brasil, diversos desafios, como o desafio de esquemas de Winograd, ainda não possuem

soluções em português brasileiro com elevada acurácia como no inglês. A solução deste desafio pode auxiliar no desenvolvimento do PLN em português brasileiro, que ainda não é tão desenvolvido como em outros idiomas.

Tendo isso em vista, insere-se no contexto também a dificuldade de trabalhar-se com a língua portuguesa. Para iniciar uma análise de PLN, é preciso um *corpus*, conceito utilizado na área para descrever o conjunto de textos em determinada língua que serve como base da análise, semelhante a um *dataset*. A grande dificuldade em relação ao trabalho com PLN em português é a obtenção de um *corpus* com tamanho adequado, quando comparado ao tamanho dos *corpora* em inglês. A Wikipédia, que em muitos casos é utilizada como *corpus* para esse tipo de estudo, chegou a um milhão de verbetes em português em 2018, enquanto que na mesma data, a versão em inglês já possuía 5,6 milhões de verbetes. [4]

1.4. Organização do Trabalho

No capítulo 2 o presente trabalho conta com aspectos conceituais, a fim de elucidar o desafio de esquemas de Winograd, exemplificando as frases ambíguas, além de conceituar o processamento de linguagem natural e apresentar o modelo escolhido. A metodologia de trabalho é descrita no capítulo 3, em que são apresentadas as abordagens utilizadas para resolução do desafio, seguidas pelas tecnologias utilizadas para implementação das soluções, no capítulo 4. Já no capítulo 5, abordam-se as especificações a serem satisfeitas e o estado da arte no desafio. No capítulo 6, são elucidados os detalhes de implementação das soluções propostas neste trabalho. Por fim, a análise dos resultados é realizada no capítulo 7, seguida pelas conclusões, no capítulo 8.

2. Aspectos Conceituais

2.1. Desafio de Esquemas de Winograd

O Desafio de Esquemas de Winograd é um desafio composto por um conjunto de frases em um formato padrão (chamados esquemas) em que existem dois substantivos de mesma característica (mesmo gênero, mesmo número) e um pronome pessoal ou pronome possessivo que se refere a um dos substantivos, mas que, gramaticalmente, poderia se referir ao outro. O desafio consiste em determinar a qual substantivo o pronome se refere, sendo que este problema é solucionado facilmente por humanos devido ao senso comum que possuem, mas sendo praticamente impossível de ser solucionado apenas por mecanismos gramaticais. Na estrutura dos esquemas há uma palavra, chamada de *palavra especial*, que, quando substituída por uma *palavra alternativa,* faz com que a nova frase também tenha sentido, porém o significado é alterado, e, então, a resposta ao desafio muda. Um exemplo é a seguinte frase, do esquema traduzido para português:

A medalha não cabe na maleta porque ela é muito grande.

O que é grande?

a. medalha

b. maleta

Neste caso, “medalha” e “maleta” são os substantivos, “ela” é o pronome e “grande” é a *palavra especial*. É do senso comum de um ser humano dizer que a resposta é **a**, visto que uma medalha muito grande provavelmente não caberia numa maleta, enquanto que uma maleta grande provavelmente seria grande o suficiente para uma medalha. Ao substituir a *palavra especial* pela *palavra alternativa* “pequena”, a frase continua fazendo sentido, mas o significado muda e a resposta passa a ser **b**. O conteúdo e temática dos esquemas não possuem regra

específica, mas, idealmente, o vocabulário utilizado deve ser suficientemente simples para que até uma criança possa responder corretamente o desafio.

Formalmente, os esquemas propostos devem seguir as seguintes regras, apresentadas em [1]:

- Um esquema deve ser facilmente resolvido por um leitor humano. Idealmente, a frase deve ser fácil o suficiente para que o leitor nem perceba a ambiguidade.
- Um esquema não deve ser solucionável com técnicas simples como restrições selecionais (restrições semânticas do pronome ou do verbo que implicam em apenas uma das opções ser correta).
- Um esquema deve ser "à prova de *Google*", ou seja, não deve haver um teste estatístico óbvio sobre o texto que resolve a ambiguidade da frase (por exemplo, no esquema apresentado anteriormente, a frequência de aparecimento, em um *corpus* textual genérico, do conjunto das palavras "maleta" e "grande" não pode ser muito diferente da frequência de aparecimento do conjunto das palavras "medalha" e "grande", pois uma grande diferença indica que o conjunto com maior frequência de aparecimento provavelmente é a resposta correta).

Este desafio é considerado um teste que demonstra que quem o responde está pensando e possui senso comum, pois, segundo essas regras, truques gramaticais e acesso a uma grande base de dados textual (que não possui respostas ao esquema) provavelmente não ajudarão em sua resolução de forma a obter um desempenho adequado. Como a construção gramatical é a mesma para ambas as respostas, alternando apenas a *palavra especial*, pode-se dizer que este desafio cria barreiras a estes tipos de abordagem. É entendido que para que se consiga um desempenho melhor do que o simples "palpite aleatório", é necessário interpretar o sentido da frase. No exemplo apresentado, é necessário ter a compreensão de que "não caber" é causado por um dos objetos ser grande ou pequeno, determinando, assim, qual é qual.

O desafio de esquemas de Winograd é relativamente recente, tendo sido publicado inicialmente em 2011, em [1]. Esta publicação inicial se refere ao desafio

em inglês, sendo que uma base com 150 esquemas é disponibilizada. É possível estendê-la a 285 esquemas considerado os esquemas com as *palavras alternativas*. Porém, um dataset de 273 esquemas - chamado de WSC273 - é o utilizado para comparação de modelos. Com o avanço do PLN, a acurácia dos algoritmos sobre esses esquemas iniciais, em inglês, já chega a aproximadamente 90%, em [5]. Porém, com isso, dúvidas surgiram sobre a real habilidade dos algoritmos terem obtido senso comum ou se há algum enviesamento nos datasets que está levando a tamanha acurácia. A fim de testar essas hipóteses para o desafio proposto em inglês, através de mais dados para teste, desenvolveu-se em [5] um novo *dataset*, chamado de WINOGRANDE, com objetivo de melhorar a escala e consistência dos esquemas, passando então dos 150 originais propostos para mais de 40 mil esquemas.

2.2. Desafios de Esquemas de Winograd em Português Brasileiro

Os esquemas de Winograd propostos originalmente em inglês foram traduzidos para português brasileiro em [2], e, visto que a ambiguidade contida na sentença pode mudar de língua para língua, como será melhor descrito a seguir, tal tradução foi feita manualmente, por pessoas falantes de inglês e português e, então, revisada. No trabalho citado, não apenas uma tradução de palavra por palavra é feita, mas também uma tradução de regras gramaticais que variam de língua para língua. Vale ressaltar que em inglês marcações de gênero não interferem na resolução do desafio. Em português, porém, muitos dos esquemas traduzidos passam a ter respostas óbvias devido a essa marcação de gênero (por exemplo, em inglês o exemplo dado anteriormente era "*The trophy doesn't fit in the brown suitcase because it's too big.*", que se traduz para "O troféu não cabe na mala marrom porque **ele** é muito grande.", sendo óbvio que "ele" se refere a troféu, devido ao gênero), e, portanto, é necessário fazer adaptações para que os esquemas mantenham as ambiguidades a serem resolvidas em português. No trabalho citado, foram traduzidos 285 esquemas de inglês para português, incluindo todos os 273 esquemas presentes no dataset WSC273, usado para comparações.

Vale ressaltar que há uma versão adicional da tradução em que os nomes próprios presentes nos esquemas são adaptados para nomes comuns em português brasileiro, enquanto que na primeira versão os nomes próprios são mantidos iguais aos esquemas em inglês. Optou-se por manter os nomes originais (nomes mais comuns em inglês) de modo a avaliar com maior precisão a qualidade do sistema de tradução automática usado na primeira abordagem de solução do projeto, que será visto a frente.

Além disso, no trabalho citado também é desenvolvido um sistema para resolver os esquemas de Winograd em português. Com isso, é utilizado como *corpus* um conjunto de palavras advindas da Wikipedia para treinamento do modelo de linguagem. O sistema se utiliza de um modelo de linguagem que atribui probabilidades a sentenças, a partir do qual se faz modificações na estrutura das frases (trocando os pronomes pelos substantivos) para que esta técnica possa ser utilizada. No exemplo “A medalha não cabe na maleta porque ela é muito grande”, o modelo testa a probabilidade das frases “A medalha não cabe na maleta porque a medalha é muito grande” e “A medalha não cabe na maleta porque a maleta é muito grande”, selecionando como resposta a frase de maior probabilidade. Essa solução para o desafio em português, apresentada em [2], a ser utilizada como ponto de partida, possui acurácia de aproximadamente 45% (no caso de treinamento através do *dataset* original utilizado, sem eventuais correções), enquanto que um ser humano atinge uma média superior a 90% no teste (os valores exatos são apresentados na seção 5.2). Este desempenho é o estado da arte atual na solução do desafio em português. Assim, espera-se desenvolver um modelo que possua um desempenho melhor neste desafio traduzido, se aproximando da acurácia de um ser humano.

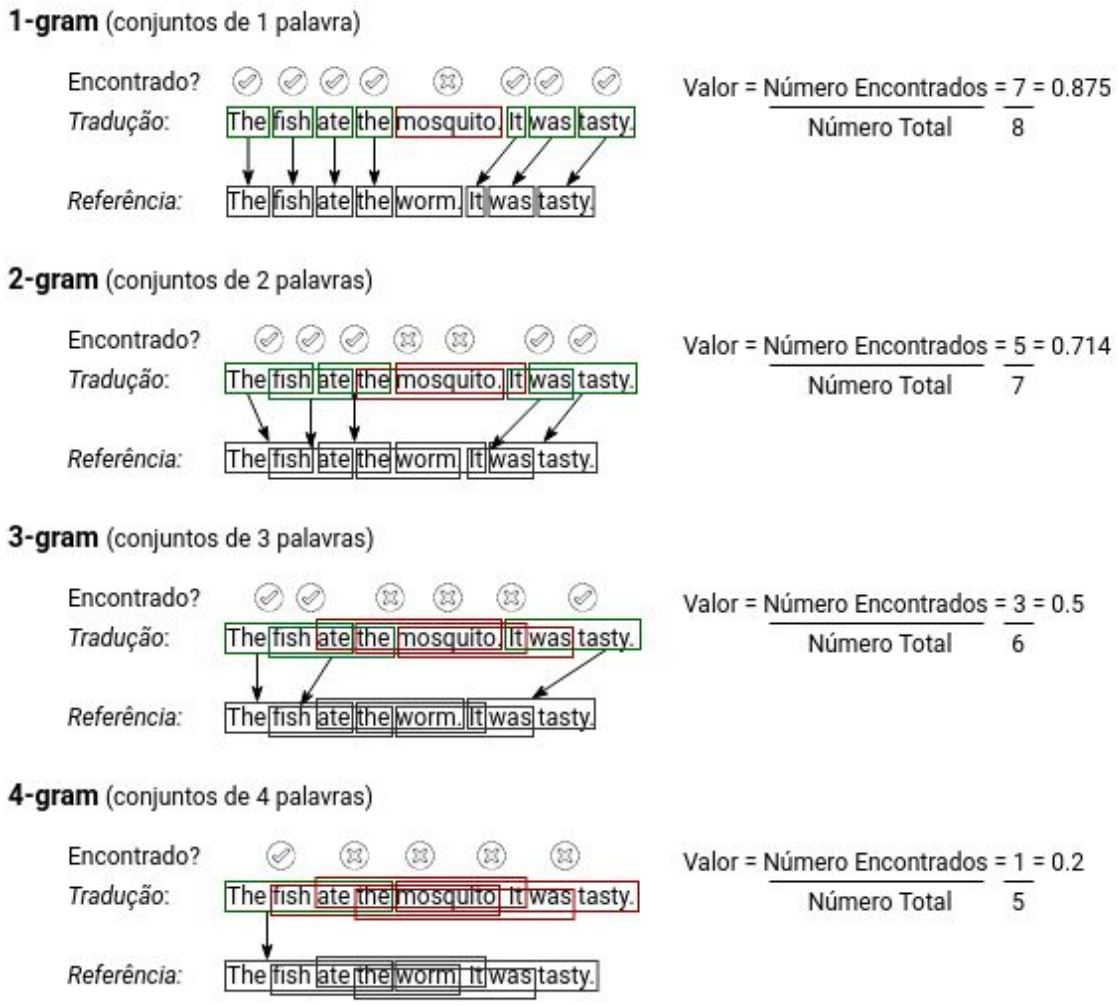
2.3. BLEU: Bilingual Evaluation Understudy

Devido à utilização, neste trabalho, de um sistema para tradução de frases entre português e inglês, que será apresentado posteriormente, é importante que haja uma forma de mensurar a qualidade e validade dessas traduções. Por isso, um conceito importante a ser elucidado é o algoritmo “*Bilingual evaluation*

understudy" (conhecido como BLEU), proposto em [6], usado para avaliar a tradução de frases entre dois idiomas. O BLEU é uma métrica que avalia uma sentença gerada a partir de um conjunto de sentenças referência, sendo 1 a pontuação que indica a correspondência exata entre a sentença gerada e alguma das referências, e 0, a situação em que a tradução proposta não possui nenhuma similaridade com nenhuma das referências. Assim, é possível avaliar quantitativamente a qualidade das traduções realizadas.

O algoritmo do BLEU funciona, simplificadamente, iterando sobre a tradução a ser classificada percorrendo-a por meio de janelas de palavras e então checando nas referências se esta janela pode ser encontrada, dando uma pontuação de 1, caso sim, e 0, caso não. Estas janelas são chamadas de n-gramas, constituídas por n palavras, que, na prática, normalmente variam de 1-grama a 4-grama. O resultado do teste é uma normalização das pontuações baseadas nos números de palavras. A figura 2.1 ilustra o cálculo do BLEU para 1 a 4-grama com um exemplo:

Figura 2.1: Cálculo do Score de BLEU para os n-gramas onde n = 1, n = 2, n = 3 e n = 4, incluindo o cálculo ponderado com pesos iguais



Fonte: Confecção Própria

Além disso, o BLEU possui formulações adicionais que visam penalizar a brevidade (ou seja, se a tradução for mais curta que a referência) e penalizar aparições adicionais de uma janela (caso n = 1, uma palavra), quando comparado ao número de aparições em cada uma das referências.

A formulação matemática do BLEU será omitida, pois ela já é implementada em bibliotecas de Linguagem Natural para diversas linguagens de programação, mas pode ser consultada em [6].

2.4 Processamento de Linguagem Natural

A comunicação e a linguagem são características intrínsecas da natureza humana sendo que, a partir delas, é possível transmitir uma infinidade de ideias e conceitos, desde os mais simples até os mais complexos. Cada palavra e cada frase falada ou escrita por um ser humano carrega muitas informações, e, por isso, a interpretação da linguagem de forma automática pode trazer grandes benefícios para sistemas computacionais.

O Processamento de Linguagem Natural (abreviado PLN, ou NLP na sigla em inglês) é um campo da Inteligência Artificial que visa possibilitar que as máquinas possam ler, entender e induzir significados a partir da linguagem natural humana. Atualmente, esse campo de estudo tem ganhado cada vez mais relevância, pois o acesso aos dados e ao poder computacional tem se tornado cada vez mais fácil e barato, possibilitando a criação de soluções na área de saúde, finanças, indústria etc.

Os casos de uso do processamento de linguagem são inúmeros, com alguns deles já bastante presentes no dia-a-dia da população com acesso a internet e a smartphones. Alguns desses casos são:

- Assistentes pessoais por voz, que reconhecem os comandos falados pelo usuário e realizam inúmeras tarefas, muitas vezes emulando uma conversa natural entre usuário e máquina;
- Sistemas de tradução com inúmeros idiomas. Atualmente, esses tradutores não traduzem as frases palavra por palavra, mas sim traduzem a frase como um todo, dentro de seu contexto, gerando traduções muito mais precisas.
- Filtros de spam em ferramentas de email, que identificam automaticamente se o conteúdo dos e-mails recebidos são indesejados;
- Sistemas que prevêem as próximas palavras de um texto, como em teclados de smartphone, que sugerem as possíveis próximas palavras a serem usadas, se adaptando aos hábitos de digitação do usuário.

Por trás destes casos de uso estão algoritmos bastante complexos que tentam, de forma simplificada, criar uma representação interna do significado das palavras dentro do contexto das frases em que estão inseridas. Como a natureza da linguagem humana é de alta complexidade, podendo o sentido de uma frase ser alterado dependendo da entonação ou da forma como foi escrita, por exemplo, o processamento de linguagem natural também é uma área de alta complexidade na computação.

2.4.1. Modelos de Linguagem

Dentro da área de Processamento de Linguagem Natural, um dos conceitos mais importantes é o conceito dos modelos de linguagem. O modelamento de linguagem é uma das tarefas do PLN cujo objetivo é prever qual a próxima palavra de uma dada frase.

Mais formalmente, dado uma sequência de palavras $x^{(1)}, x^{(2)}, ..., x^{(t)}$ um modelo de linguagem calcula a distribuição de probabilidade da próxima palavra $x^{(t+1)}$:

$$P(x^{(t+1)}|x^{(t)}, ..., x^{(2)}, x^{(1)})$$

Os modelos de linguagem podem também ser pensados como sistemas que atribuem uma probabilidade a um trecho de texto. Por exemplo, dado um texto composto pelas palavras $x^{(1)}, x^{(2)}, ..., x^{(T)}$, a probabilidade deste texto, de acordo com o modelo de linguagem é:

$$P(x^{(1)}, x^{(2)}, ..., x^{(T)}) = P(x^{(1)}) \times P(x^{(2)}|x^{(1)}) \times ... \times P(x^{(T)}|x^{(T-1)}, ..., x^{(1)}) =$$

$$= \prod_{t=1}^{T} P(x^{(t)}|x^{(t-1)}, ..., x^{(1)})$$

Os modelos de linguagem têm diversas aplicações como, por exemplo, na predição da próxima palavra no teclado de *smartphones*, como citado anteriormente, ou também na previsão de pesquisas em ferramentas de busca, como o *Google*. Modelos de aprendizado de máquina que fazem o modelamento de linguagem conseguem, a partir desta tarefa, aprender representações internas dos conceitos e significados da língua, tornando esses modelos aplicáveis a diversas tarefas de PLN.

2.4.2. Transfer Learning

Algoritmos convencionais de aprendizado de máquina normalmente são projetados para que funcionem isoladamente, resolvendo tarefas específicas para as quais foram designados. Neste caso, para cada nova tarefa ou para alterações na natureza da tarefa, grandes modificações devem ser feitas ou, até mesmo, uma reconstrução do algoritmo deve ser realizada, adaptando-o a esta nova tarefa. *Transfer Learning* (Transferência de Aprendizagem, em tradução livre) é a ideia de superar o isolamento entre sistemas de aprendizado de máquina, utilizando o conhecimento adquirido em uma tarefa para resolver tarefas relacionadas.

Em diversas tarefas do campo do Processamento de Linguagem Natural, um dos maiores desafios é a falta de um conjunto de dados de treinamento com as características específicas da tarefa que seja grande o suficiente para o treinamento de modelos. No caso da tarefa a ser resolvida neste trabalho, por exemplo, as frases que possuem a estrutura específica apresentada pelo desafio de Esquemas de Winograd não passam de alguns milhares, com esses milhares apresentados em [5].

Porém, os modelos modernos de PLN baseados em aprendizado profundo (*deep learning*) tem um elevado aumento no desempenho quando treinados em milhões, ou até bilhões, de exemplos textuais. Para resolver este problema, a comunidade pesquisadora no campo de PLN criou uma variedade de técnicas que possibilitam o treinamento de modelos em enormes *datasets* textuais (normalmente de enciclopédias ou de bibliotecas de livros), aprendendo uma representação geral da língua. Este tipo de treinamento, chamado pré-treinamento (*pre-training*) é feito, como já citado, em milhões de exemplos textuais, demandando muito tempo e poder computacional.

A ideia então é que o modelo pré-treinado seja disponibilizado para uso de modo que o processo de pré-treinamento não precise ser repetido pelo usuário. Este modelo pode então passar pelo processo de ajuste fino (*fine-tuning*), uma segunda etapa de treinamento, em *datasets* menores, específicos da tarefa para qual o modelo irá ser utilizado para resolver. Esta abordagem, na maior parte das vezes, apresenta resultados muito melhores do que o treinamento apenas no

dataset específico da tarefa, pois, nesta abordagem, primeiro o modelo aprende uma representação geral da língua a partir de uma grande quantidade de dados para então adaptar essa representação aos dados específicos da tarefa.

O *Transfer Learning* é um conceito também muito utilizado em tarefas relacionadas à visão computacional, na qual grandes treinamentos são feitos em bancos de imagens, de modo que os modelos possam entender de forma geral as imagens do mundo (objetos, paisagens, pessoas etc), e então o ajuste fino é feito em tarefas específicas, como reconhecimento facial, por exemplo.

2.5. BERT: Bidirectional Encoder Representations from Transformers

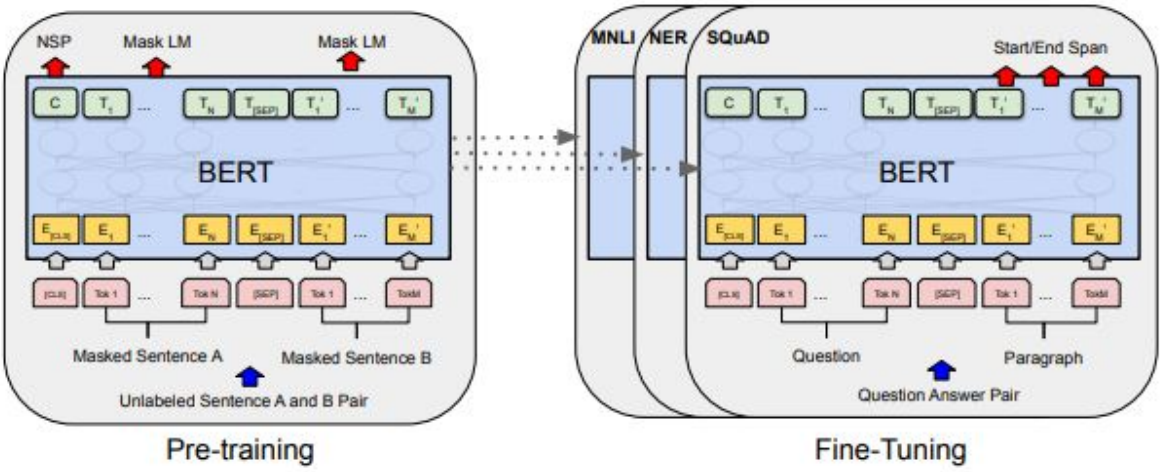
As abordagens utilizadas neste trabalho contemplam a escolha de um algoritmo ou modelo a ser utilizado para a solução do Desafio de Esquemas de Winograd. O modelo que se mostrou mais adequado para a resolução do desafio é o BERT (*Bidirectional Encoder Representation from Transformers*). As motivações para essa escolha serão elucidadas ao longo desta seção.

O BERT, apresentado em [7], é um modelo focado em PLN baseado em *deep learning* (aprendizado profundo), criado por pesquisadores do *Google AI Language*, que atingiu o estado da arte em diversas aplicações de processamento de linguagem natural, como *Question Answering* (Respostas a perguntas - benchmark SQuAD v1.1), *Natural Language Inference* (Inferência de Linguagem Natural - benchmark MLNI), dentre outros, como citado em [7]. O BERT normalmente é chamado de modelo de linguagem, mas sendo mais rigoroso, é uma arquitetura de aprendizado de máquina que inclui técnicas de treinamento, sendo base para modelos que são aplicados em diversas tarefas de PLN. Ainda assim, neste trabalho ele é referido como modelo de linguagem, ficando implícita esta rigorosidade. Uma de suas aplicações, a ser vista a frente (*Masked-Language Modeling*) é similar a de um modelo de linguagem, porém em uma versão bidirecional.

O modelo foi projetado de modo a ser pré-treinado em representações bidirecionais profundas a partir de textos não rotulados pelo condicionamento

conjunto do contexto esquerdo e direito das palavras em todas as suas camadas. A partir deste pré-treinamento o modelo pode receber ajuste-fino através de uma camada de saída (*output layer*) adicional, de modo a chegar nos resultados de estado da arte supracitados, sem que sejam necessárias grandes mudanças na arquitetura do modelo baseadas na aplicação. Na figura 2.2, de [7], pode-se notar que a arquitetura é semelhante para o pré-treinamento (à esquerda) e para o ajuste-fino (à direita)

Figura 2.2: Arquitetura dos procedimentos de pré-treinamento e ajuste fino



Fonte: Obtido na referência [7].

Este treinamento bidirecional (ou, sendo mais rigoroso, não-direcional), onde toda a sequência textual é lida de uma vez, é a grande inovação do BERT, quando comparado a esforços prévios na área de PLN que, na maior parte das vezes, realizavam o processamento da sequência textual da esquerda para a direita, ou então, usavam uma combinação do treinamento da esquerda para a direita e da direita para esquerda (tentando emular um treinamento bidirecional). Esta inovação permite que o modelo considere o contexto completo de uma palavra em uma frase, sendo esta percepção de contexto crucial para o melhor desempenho em algumas tarefas do campo de PLN.

Um dos pontos que fazem o BERT ser adequado para este trabalho é a sua disponibilidade gratuita em diversas versões. Após sua publicação, modelos pré-treinados foram disponibilizados para download, não sendo necessária a

execução do pré-treinamento, bastando apenas o ajuste fino. Dentre os modelos pré-treinados disponibilizados estão duas versões de modelo com treinamento em um corpus em língua inglesa (BERT Base e BERT Large) e uma versão treinada em um corpus multi-idiomas, disponibilizado apenas na arquitetura BERT Base. Este segundo modelo é chamado de BERT-Multilingual. Há ainda uma versão pré-treinada em português brasileiro (com as duas arquiteturas disponíveis), que foi disponibilizada em [8].

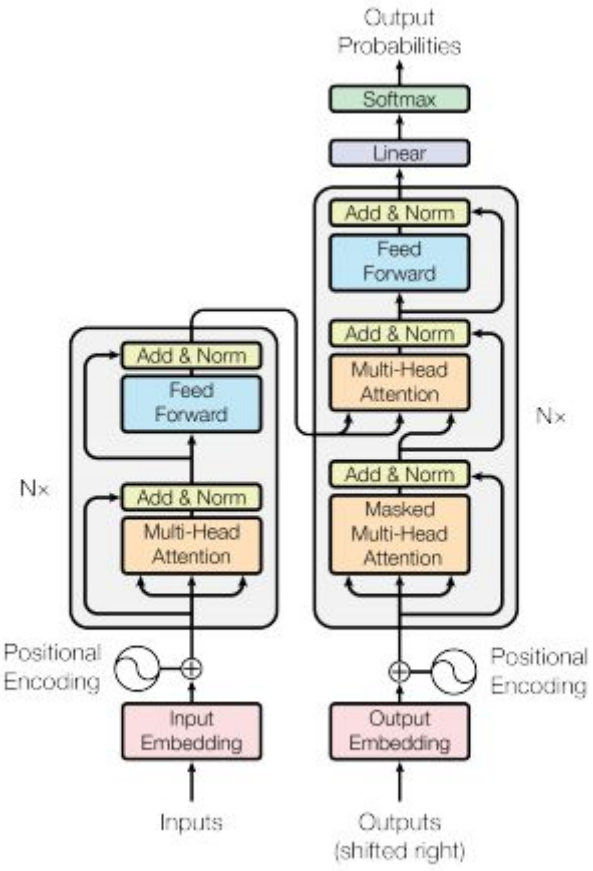
2.5.1. Arquitetura do BERT

De maneira geral, o BERT é um conjunto de codificadores treinados, advindos da arquitetura de um *transformer* - mecanismo que usa o conceito de *atenção,* de [9], que aprende relações contextuais entre palavras em um texto. Os dois modelos disponibilizados, já citados, são:

- BERT Base, que possui 12 camadas (blocos de *Transformers*), 12 *attention heads* e 110 milhões de parâmetros.
- BERT Large, que possui 24 camadas (blocos de *Transformers*), 16 *attention heads* e 340 milhões de parâmetros. Este é o modelo com o qual se obteve resultados que alcançaram o estado da arte em tarefas de PLN.

Um *transformer* tem uma arquitetura composta por um codificador, que lê a entrada de texto, e um decodificador, que produz uma previsão para a tarefa, como mostrado na figura 2.3 abaixo:

Figura 2.3: Arquitetura de um *transformer*, com o codificador representado à esquerda e o decodificador à direita.



Fonte: Obtido na referência [9].

Um ponto importante da arquitetura de um transformer é a camada de *atenção*, citada na figura 2.3 como *"Multi-Head Attention"* (sendo chamada de "multi-head" por representar os diferentes cálculos da *atenção*, iniciadas com diferentes parâmetros), sendo este ponto bastante relevante para o uso do BERT neste trabalho. O método da *atenção* (também chamado de *Self-Attention*) é um método que detecta relações entre as palavras de uma frase. Seja a seguinte frase um exemplo que passa pelo *transformer*:

O animal parou de correr pois ele estava cansado.

Na camada de "*Multi-Head Attention"* a *atenção* é calculada em diferentes subespaços de representação (diferentes parâmetros de cálculos), permitindo a detecção de diferentes relações entre as palavras. Por exemplo, o pronome "ele" pode ser identificado como tendo relação com as palavras "o" e "animal", pois se

refere a essas palavras. Ou então, em outro subespaço de representação, esse mecanismo pode identificar a relação entre o pronome “ele” e o seu adjetivo “cansado”. O embasamento matemático para o cálculo da *atenção* é apresentado em [9]. Este mecanismo é de extrema importância para que o BERT possa aprender o contexto da frase e a conexão entre as palavras.

Como o objetivo do BERT é ter o funcionamento similar a de um modelo de linguagem, só o codificador advindo da arquitetura do *transformer* é utilizado, não sendo necessário o decodificador, visto que essa arquitetura foi proposta, inicialmente, para problemas de tradução (sequência para sequência), em que é necessário a produção de uma sequência de saída (a tradução de uma dada frase), sendo esta tarefa realizada pelo decodificador, não necessário para o propósito do BERT.

2.5.2. Pré-treinamento do modelo

O pré-treinamento realizado no BERT é auto-supervisionado, de modo que o treinamento é feito em textos “crus”, sem que seres humanos tenham os rotulado, usando um processo automático de geração das entradas e rótulos destes textos. Essa auto-suficiência no pré-treinamento é essencial para que este seja realizado em enormes bases de dados textuais.

O pré-treinamento é feito em duas tarefas:

- *Masked Language Modeling* (MLM - Modelamento de Linguagem Mascarada): Nesta tarefa, dada uma sentença, o modelo mascara (insere o token [MASK]) 15% das palavras da frase e executa o modelo sobre a sentença, tentando prever quais são as palavras mascaradas. Em modelos de linguagem tradicionais, normalmente o pré-treinamento é feito prevendo a próxima palavra de uma frase. No MLM, porém, as palavras mascaradas estão dentro do contexto da sentença, sendo que o modelo usa as palavras anteriores e posteriores ao [MASK] para prever a palavra mascarada. Desta forma, o modelo aprende a representação bidirecional da sentença.
- *Next sentence prediction* (NSP - Predição da Próxima Sentença): Na segunda tarefa do pré-treinamento do BERT, o modelo seleciona duas sentenças, também mascaradas. No texto original, essas sentenças podem

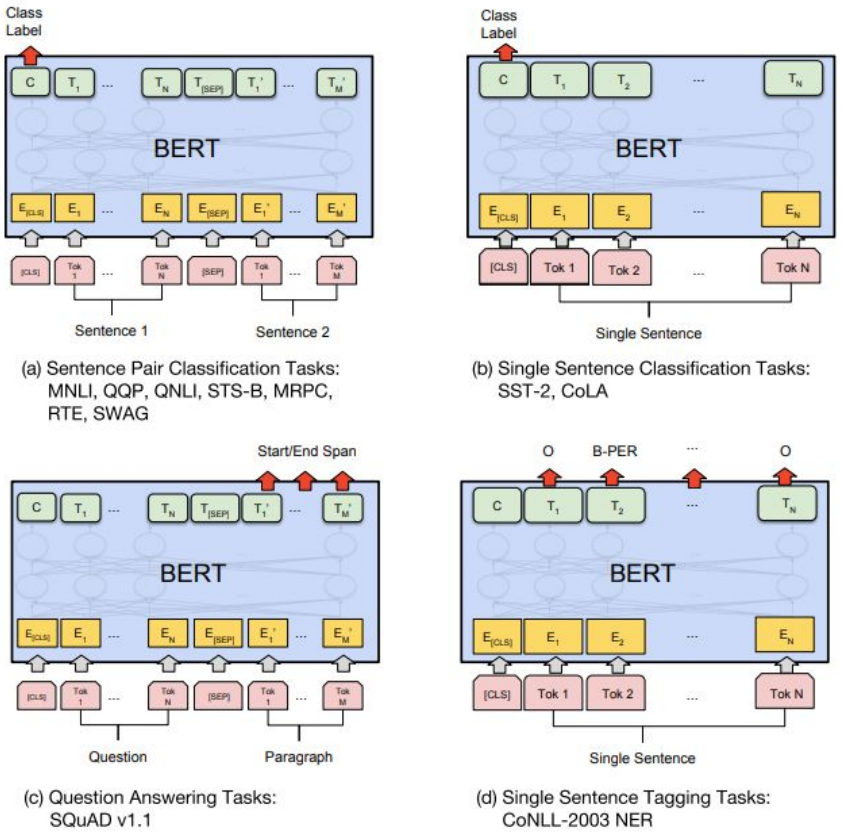
ser sentenças consecutivas ou sentenças desconexas. O objetivo desta tarefa é que o modelo preveja se as sentenças são consecutivas ou não.

Estas duas tarefas de pré-treinamento fazem com que o modelo ajuste seus parâmetros, a partir de seus erros e acertos, para que seja criada uma representação interna do idioma no qual as tarefas foram executadas. O modelo pode então ser treinado por meio de ajuste fino para realizar tarefas específicas do processamento de linguagem natural.

2.5.3. Casos de Uso do BERT

A partir deste pré-treinamento, o BERT pode ser utilizado em uma variedade de outras tarefas, bastando realizar o ajuste-fino. Algumas destas tarefas, cujas arquiteturas podem ser vistas na figura 2.4, de [7], são: classificação de pares de sentenças, classificação de sentenças únicas, respostas a perguntas, marcação de sentenças únicas, dentre outras. Em algumas outras tarefas de processamento de linguagem, porém, o pré-treinamento do BERT não traz grande benefícios e outros modelos são recomendados, como, por exemplo, na tarefa de geração de texto.

Figura 2.4: Arquiteturas do BERT para ajuste fino em diferentes tarefas.



Fonte: Obtido na referência [7].

Além das tarefas já citadas, o BERT também pode ser utilizado nas próprias tarefas usadas para o seu pré-treinamento. Para a resolução do desafio de esquemas de Winograd, a tarefa de *Masked Language Modeling* é essencial. Fazendo alguns ajustes nos esquemas, de modo a mascarar o pronome ambíguo, pode-se usar o modelo para calcular a probabilidade dos dois candidatos serem a substituição adequada. Aquele com a maior probabilidade é então selecionado como melhor candidato à resposta do esquema. No exemplo anteriormente citado, o esquema é modificado para “A medalha não cabe na maleta porque [MASK] é muito grande”, e calcula-se a probabilidade das palavras “medalha” e “maleta” substituírem o campo [MASK], escolhendo a resposta de maior probabilidade.

Assim como nas outras tarefas, para que o modelo tenha melhor desempenho, é importante que seja feito um treinamento de ajuste fino, de modo a possibilitar que o modelo adeque seus parâmetros à tarefa.

3. Metodologia do Trabalho

A metodologia utilizada no presente trabalho é composta por diferentes abordagens para a solução do desafio de esquemas de Winograd para que, ao final das mesmas, seja possível fazer uma comparação entre elas a fim de obter conclusões sobre qual abordagem é mais satisfatória para a solução do desafio e sobre o diferente comportamento e caso de uso das abordagens.

Uma das abordagens, considerada como um bom passo inicial, é a criação de um *baseline* baseado nas soluções em inglês do desafio de Winograd. Dado que as soluções em inglês são mais satisfatórias que aquelas desenvolvidas originalmente em português, uma opção é traduzir o esquema a ser solucionado de português para inglês e então rodar o modelo de linguagem que foi treinado com um *corpus* em inglês, com e sem ajuste fino em uma *dataset* similar ao WSC. Para que a aplicação dessa abordagem seja possível, é desenvolvido um *script,* disponível no repositório do Github criado para o projeto[1], que realiza a tradução automaticamente (através de uma API dedicada a essa finalidade) e que então envia a frase traduzida para uma instância do algoritmo em inglês. Esta primeira abordagem visa entender a viabilidade de transferência de conhecimento entre idiomas em desafios de PLN. Neste caso, traduz-se o desafio em português para um idioma com melhor desempenho no desafio (no caso, em inglês), de modo a aproveitar-se desses melhores resultados.

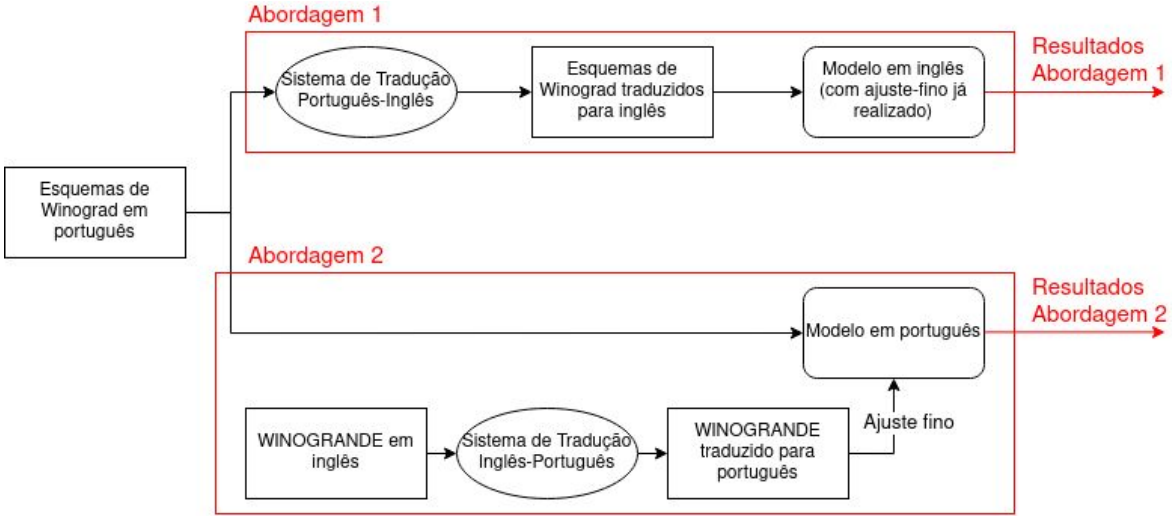
Uma segunda abordagem é a utilização de um modelo de linguagem treinado em um *corpus* em português brasileiro para a solução do desafio traduzido. Para alcançar um desempenho comparável à primeira abordagem, é necessário realizar o ajuste fino deste modelo, sendo uma das opções traduzir os mais de 40 mil esquemas do WINOGRANDE, proposto em [5], usando o *dataset* resultante para ajuste fino e, com o modelo treinado, aplicá-lo ao desafio de esquemas de Winograd em português brasileiro. Esta abordagem visa analisar a possibilidade de utilização de modelos em português para a resolução do desafio e

1. Repositório oficial do projeto. Disponível em <https://github.com/thaapontes/tcc_project> Acesso em: 16 nov. 2020.

avaliar a eficiência no uso do WINOGRANDE traduzido como *dataset* para ajuste fino.

Após a implementação e execução das abordagens propostas, são feitas análises e comparações a partir dos resultados obtidos. A figura 3.1 demonstra, simplificadamente, as duas abordagens utilizadas:

Figura 3.1: Fluxograma com as duas abordagens para resolução do WSC em português utilizadas neste trabalho



Fonte: Confecção própria

4. Tecnologias Utilizadas

Para ambas as abordagens propostas, faz-se necessário o uso de uma forma de traduzir esquemas de português para inglês e vice-versa. Na primeira abordagem, traduz-se os esquemas do *dataset* WSC273 de português para inglês, executando o modelo de linguagem na língua inglesa. A segunda abordagem utiliza tanto os esquemas como o modelo em português porém, para o ajuste fino do modelo é necessária a tradução, de inglês para português, do *dataset* WINOGRANDE, de [5].

Desta forma, devido à importância da tradução para este trabalho, este foi um dos primeiros pontos a serem elucidados. De forma a resolver o problema de forma automatizada, buscaram-se e testaram-se diversas APIs grátis para tradução, disponíveis em *Python*, sendo a que se mostrou mais adequada para o objetivo foi a API do *Google Cloud*, que utiliza um algoritmo semelhante ao do *Google* Tradutor, com algumas diferenças (neste caso, a utilização não é gratuita, mas o *Google* fornece um valor para testes na plataforma, suficiente para este trabalho).

Na primeira abordagem, buscou-se o arquivo que continha os esquemas em língua portuguesa e percebeu-se que o mesmo se encontrava em formato HTML, mais difícil de ser trabalhado com a tecnologia *Python* escolhida, então desenvolveu-se um *script* para passá-lo ao formato .json. Na segunda abordagem, os esquemas de [5] estavam em formato .jsonl, e também foi necessário criar um *script* no repositório do projeto[1], usando *Python* para conversão ao .json, possibilitando a execução da tradução.

Para aplicação do algoritmo BLEU, descrito anteriormente, nas frases cuja tradução foi avaliada, utilizou-se uma biblioteca em *Python*, chamada “NLTK” (*Python Natural Language Toolkit library*), que possui uma implementação deste algoritmo.

Para a execução e, principalmente, o treinamento de algoritmos que trabalham com linguagem natural, é recomendável a utilização de GPUs (*Graphics Processing Unit*), dado que é um processo no qual a paralelização das tarefas

acelera significativamente o processamento. Dada a limitação do *hardware* pessoal e a impossibilidade de locomoção à universidade devido às restrições trazidas pela pandemia ocorrida em 2020, foi necessário ter soluções alternativas para a execução e treinamento dos modelos em GPUs. Uma forma possível para utilização gratuita de GPUs é a plataforma *Google Colaboratory* (abreviada como *Google Colab*), um espaço para programação baseado em *Python* que permite ao usuário a conexão gratuita com GPUs do *Google*. Por ser gratuito, o *hardware* alocado pode variar dependendo do momento da conexão, podendo oferecer desempenhos variáveis. Além disso, a ferramenta possui algumas limitações como tempo máximo de execução de 12 horas, o que acaba limitando a capacidade de treinamento prolongado de modelos. O uso desta ferramenta foi essencial neste trabalho, pois permitiu a colaboração remota e a execução dos testes e treinamentos independente de *hardwares* específicos.

Toda a execução dos modelos do BERT foi realizada nesta ferramenta a partir de uma solução em código aberto, de [10], com as modificações necessárias, sendo os detalhes da implementação elucidados no capítulo 6. A tecnologia utilizada foi o Python, com o uso de diversas bibliotecas, sendo a principal a *PyTorch*, uma biblioteca de código aberto focada em *Machine Learning*.

5. Especificação

Neste capítulo foram entendidos os resultados aceitáveis e a serem buscados em relação ao sistema de tradução e em relação às soluções existentes para o Desafio de Winograd tanto em inglês, como em português brasileiro.

Em relação ao Desafio de Esquemas de Winograd, foram obtidos diversos resultados de trabalhos em língua inglesa, que, como dito anteriormente, possui algoritmos de processamento de linguagem natural mais desenvolvidos. Já em português brasileiro, poucos trabalhos existem sobre o desafio, tendo sido usado como referência neste documento o trabalho [2], que é o estado da arte atualmente.

Em seguida, observaram-se os resultados alcançados por cada trabalho, como descrito na seção 5.2, e, para especificar as condições de desempenho satisfatório no presente trabalho, ponderou-se a acurácia obtida no trabalho em português e nos trabalhos com o desafio em inglês.

5.1. BLEU

Em relação ao sistema de tradução desenvolvido, cuja avaliação de seu desempenho é feito pelo algoritmo de BLEU, citado na seção 2.3, é importante comparar o desempenho com referências, para validar o sistema desenvolvido. Em [6], podemos obter resultados de referência de uma tradução de chinês para inglês, com as frases de referência sendo feitas por tradutores profissionais. Os resultados são apresentados abaixo, na figura 5.1, onde H1 e H2 são seres humanos que participaram do teste, sendo H1 uma pessoa não nativa nem em chinês nem em inglês, enquanto que H2 é um nativo em inglês. S1, S2 e S3 são sistemas comerciais de tradução (vale ressaltar que [6] foi publicado em 2002 e os sistemas comerciais de tradução tiveram melhoras significativas até o momento atual).

Figura 5.1: Resultados de referência do BLEU]

S1	S2	S3	H1	H2
0.0527	0.0829	0.0930	0.1934	0.2571

Fonte: Obtido na referência [6]

Estes resultados são uma média simples entre o resultado do BLEU nos quatro n-gramas citados (1, 2, 3 e 4-grama). Vale ressaltar que este é o resultado médio do desempenho em 500 sentenças, utilizando duas referências no cálculo. É importante salientar que a comparação entre resultados com número diferente de referências deve ser cauteloso, pois o número de referências por frase é diretamente proporcional ao resultado do BLEU.

5.2. Especificação no Desafio de Esquemas de Winograd

Como dito em 1.1., o objetivo deste trabalho é o desenvolvimento de uma solução estado da arte para o desafio de esquemas de Winograd em português. No trabalho em que houve a tradução dos esquemas para português [2], é também desenvolvido uma solução para o problema a ser usada como *baseline* para novas soluções, que é o atual estado da arte no desafio.

Para o conjunto de dados original e completo, em português, com os 285 esquemas traduzidos, obteve-se uma acurácia de 45,13%, chegando até 45,49% no caso em que ocorrem correções manuais e são usados nomes em português. Vale ressaltar que no presente trabalho será usado o *dataset* WSC273 para a avaliação, pois é o padrão de comparação entre algoritmos focados na resolução do WSC. Porém, como a diferença é de apenas 12 esquemas, os resultados de [2] sem modificações serão utilizados para comparação.

Em língua inglesa, os resultados no desafio de esquemas de Winograd são os mais satisfatórios, dado o avanço dos modelos de linguagem e do processamento de linguagem natural neste idioma. Portanto, é válido apresentar os resultados em inglês para que seja possível realizar uma comparação entre o desempenho no desafio em ambos os idiomas. Vale ressaltar que todos os resultados apresentados se referem ao WSC273 como *dataset* de avaliação.

Em [10], obtém-se uma acurácia de 61,9% com o BERT-Large em inglês. Nesse mesmo trabalho, com o ajuste fino sobre o BERT-Large em um *dataset* com frases com ambiguidades pronominais similares ao WSC, denotado por WSCR (chamado de *Definite Pronoun Resolution dataset)*, chega-se a uma acurácia de

72,5%. Este modelo é de grande relevância para este trabalho pois possui código aberto, disponível no Github do autor, em [10], o que possibilita a reutilização do código para as abordagens citadas anteriormente.

Em [5], obtém-se uma acurácia de 90,1%, utilizando uma modificação do BERT conhecida como RoBERTa, com ajuste-fino no *dataset* WINOGRANDE, apresentado no mesmo trabalho. Esse trabalho é considerado o estado da arte, atualmente. Como comparativo, nesse mesmo trabalho é medido o desempenho humano no desafio, sendo de 96,5%.

Portanto, dado o objetivo deste projeto que é o desenvolvimento de uma solução estado da arte para o desafio de esquemas de Winograd em português brasileiro, sendo essa satisfatoriedade baseada na superação de soluções já desenvolvidas em português brasileiro, pode-se dizer que resultados acima de 45,5% já são satisfatórios. Como as soluções em inglês já se aproximam do desempenho humano, alcançar um desempenho comparável à solução de [10], disponibilizada em código aberto, com acurácia de 72,5% já seria bastante satisfatório. Um solução comparável à solução de [5], obtendo uma acurácia de 90,1% seria excepcional.

6. Implementação

6.1. Abordagem 1: Tradução dos Esquemas de Português para Inglês e Execução do Modelo de Linguagem em Inglês

A abordagem inicial escolhida foi a de traduzir os esquemas de Winograd em português, de [2], para o inglês e utilizar um modelo de linguagem pré-treinado e com ajuste-fino em inglês. Para isso, é necessário deixar os esquemas em português em um formato adequado para que os mesmos possam ser traduzidos e, em seguida, realizar a tradução. Com o arquivo de esquemas traduzidos em mãos, é importante metrificar quantitativamente a qualidade da tradução. Optou-se por utilizar o *score* de BLEU, conceitualmente descrito na seção 2.3. Não obstante, algumas modificações no arquivo traduzido podem ser feitas para adequar o arquivo de entrada ao modelo de linguagem já pré-treinado.

Desta forma, dependendo dos resultados do algoritmo e de uma avaliação manual, algumas correções serão necessárias. O próximo passo é implementar o modelo de linguagem e então executá-lo, avaliando os resultados obtidos.

6.1.1 Tradução dos esquemas para o inglês

O primeiro passo da primeira abordagem descrita é a tradução dos esquemas em língua portuguesa para a inglesa. Para tanto, visto que os esquemas disponibilizados por [2], em português, estavam em formato HTML, fez-se necessário convertê-los para um formato mais fácil de manipular, o *.json*. Sendo assim, desenvolveu-se um *script*, disponível no repositório do projeto[1], usando Python para realizar tal conversão entre formatos dos arquivos. Todos os *scripts* em Python desenvolvidos para adaptação de dados, tradução e para cálculo do BLEU estão disponíveis no repositório do projeto[1].

Com o arquivo em formato *.json*, foi possível implementar a tradução dos esquemas utilizando a API de tradução disponível no *Google Cloud*, instanciada e executada através de código *Python*. O arquivo obtido da tradução mantém a mesma organização dos esquemas em português sendo os esquemas compostos

pela sentença ambígua, o trecho que contém a ambiguidade (chamada de *snippet*), o pronome, as duas substituições possíveis e qual delas é a resposta certa (A ou B). Um exemplo do esquema traduzido pode ser visto abaixo:


```
{
    "schema": "The councilors refused authorization to protesters because
    they feared violence.",
    "snippet": "they feared violence.",
    "pronoun": "they",
    "correct_answer": "A",
    "substitution_a": "The councilors",
    "substitution_b": "The protesters",
    "translated": true,
    "pronoun_from_snippet": "they"
}
```


Além dos campos citados, pode-se notar que outros dois campos foram adicionados à tradução sendo eles o campo *translated*, que diferencia os esquemas traduzidos (*translated = true)* dos esquemas originais em português (*translated = false*). O outro campo é o *pronoun_from_snippet* que foi utilizado para comparar a tradução do pronome por si só (sem contexto) e o pronome traduzido dentro do *snippet* (trazendo um pouco de contexto através das palavras adicionais do *snippet*) de modo a entender as diferenças de tradução.

Vale ressaltar que o arquivo utilizado para fazer a tradução é o proposto pelo trabalho em [2], que possui 285 esquemas adaptados para o português (as 12 frases adicionais, que não compõe o WSC273, também foram traduzidas para a análise a partir do BLEU, mas não foram utilizadas nas avaliações). Após realizada a tradução, notou-se alguns erros de tradução em certos esquemas. Um dos erros é relativo à tradução dos substantivos, que algumas vezes ocorreu de forma distinta para a sentença ambígua e o campo de substituição, como pode ser visto no exemplo abaixo:

a. Esquema: The medal does not fit in the **briefcase** because it is too big.
    i. Substituição a: the medal
    ii. Substituição b: the **suitcase**

Outro erro de tradução encontrado é relacionado ao pronome ambíguo que foi traduzido de forma distinta dentro da sentença e no campo específico de pronome, como visto a seguir:

a. Esquema: The big ball crossed the table because **it** was made of steel.
    i. Pronome: **she**

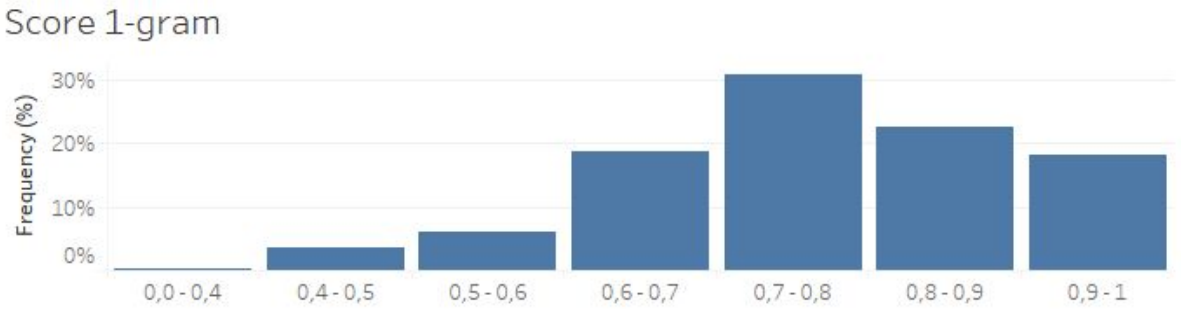
Além dos erros supracitados, que são referentes à estrutura dos esquemas e as diferentes traduções da API baseadas no contexto, deseja-se também avaliar quantitativamente a qualidade da tradução como citado anteriormente e, para isso, utilizou-se o algoritmo BLEU, tema da próxima subseção.

6.1.2 Metrificando a tradução

Para metrificar o quão satisfatória foi a tradução dos esquemas, implementou-se o *score* de BLEU, cuja descrição teórica se encontra em 2.3. Para isso, utilizou-se uma biblioteca em *Python*, como mencionado no capítulo 4. Este algoritmo de metrificação de traduções pode ser utilizado, neste caso pois os esquemas originais em inglês são utilizados como a referência no cálculo. Em traduções em que não há ao menos uma tradução de referência, não é possível utilizar o BLEU.

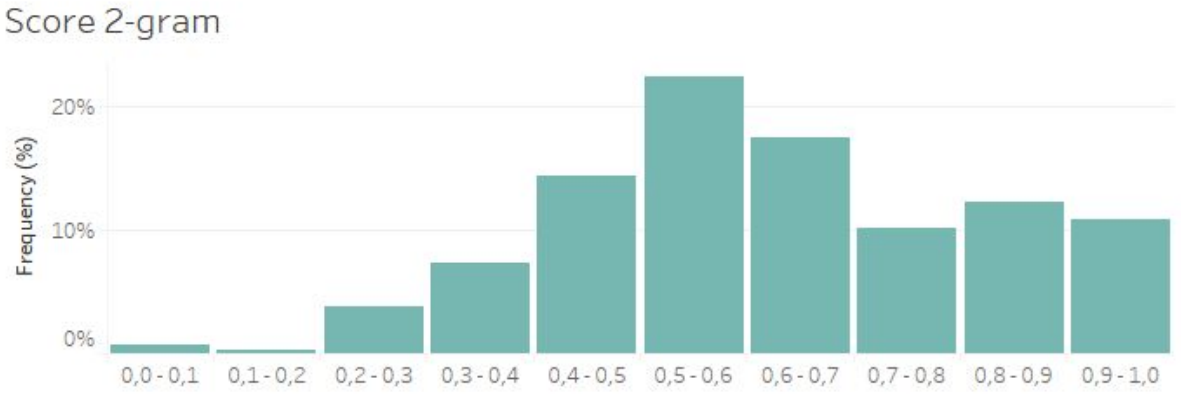
O modelo de tradução desenvolvido, quando utilizado para traduzir os esquemas originais do desafio de esquemas de Winograd, utilizando os dados de [1] como referência para tradução e os esquemas de [2], em português, que passam pela tradução, teve as seguintes distribuições de frequência para cada n-gram:

Figura 6.1: Resultados do BLEU para tradução dos esquemas, score 1-gram.



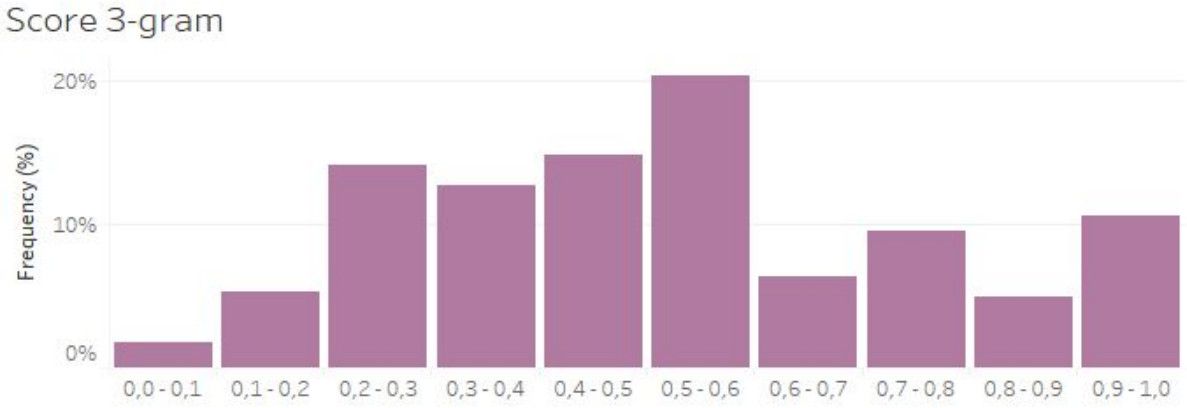
Fonte: Confecção própria.

Figura 6.2: Resultados do BLEU para tradução dos esquemas, score 2-gram.



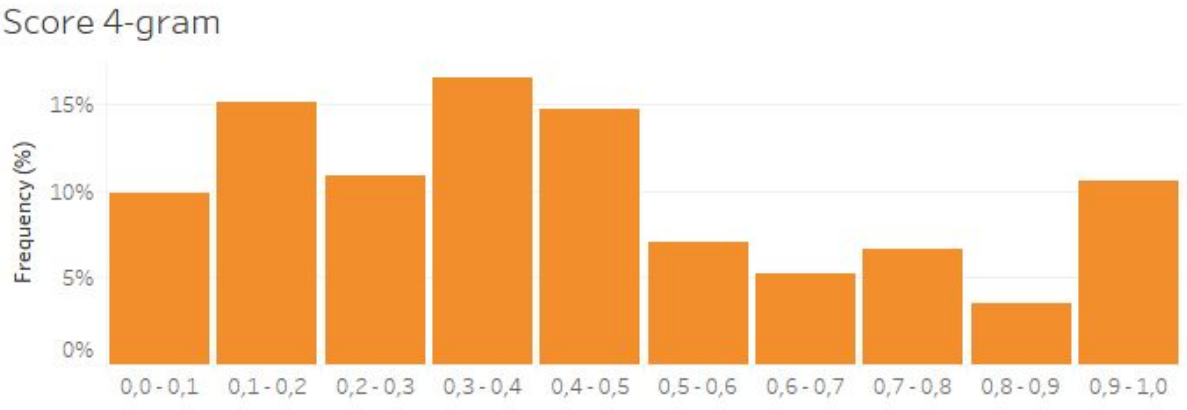
Fonte: Confecção própria.

Figura 6.3: Resultados do BLEU para tradução dos esquemas, score 3-gram.



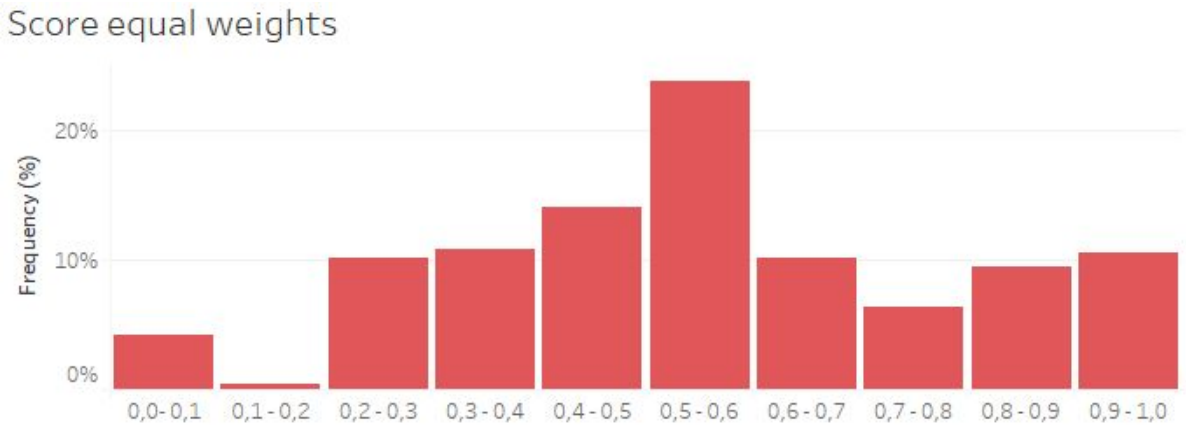
Fonte: Confecção própria.

Figura 6.4: Resultados do BLEU para tradução dos esquemas, score 4-gram.



Fonte: Confecção própria.

Figura 6.5: Resultados do BLEU para tradução dos esquemas, score “equal-weights”.



Fonte: Confecção própria.

A estatística descritiva que optou-se por utilizar a fim de comparar com a referência apresentada no capítulo 5, de especificações, foi a média, que obteve o seguinte resultado (apresentado na tabela 6.1), sendo “equal-weights” (pesos iguais, seguindo a nomenclatura de [6]) o valor a ser comparado com a referência (média de 1, 2, 3 e 4-grama).

Tabela 6.1: Resultados do BLEU para tradução dos esquemas em português.

n-gramas	1-gram	2-gram	3-gram	4-gram	equal weights
Média	0,77	0,62	0,51	0,43	0,55

Além disso, também calculou-se a porcentagem de sentenças que foram traduzidas exatamente iguais à referência, na Tabela 6.2, de modo a entender a prevalência das traduções exatamente iguais.

Tabela 6.2: Porcentagem de sentenças idênticas ao dataset original.

% Sentenças idênticas	9,824%

Comparando os resultados apresentados com os dados de referência da figura 5.1, pode-se concluir que o sistema de tradução desenvolvido se adequa às especificações propostas, validando a sua utilização neste projeto.

Entretanto, apesar de os resultados terem sido adequados para a especificação proposta, ainda existem algumas falhas na tradução, como exemplos do subtópico anterior, em que o pronome é traduzido diferente na sentença e no campo ‘pronome’, ou quando os substantivos são traduzidos de forma distinta na sentença e no campo de ‘substituição’. A análise e ajustes realizados em sentenças com tais erros de tradução são melhor detalhados durante a avaliação dos resultados.

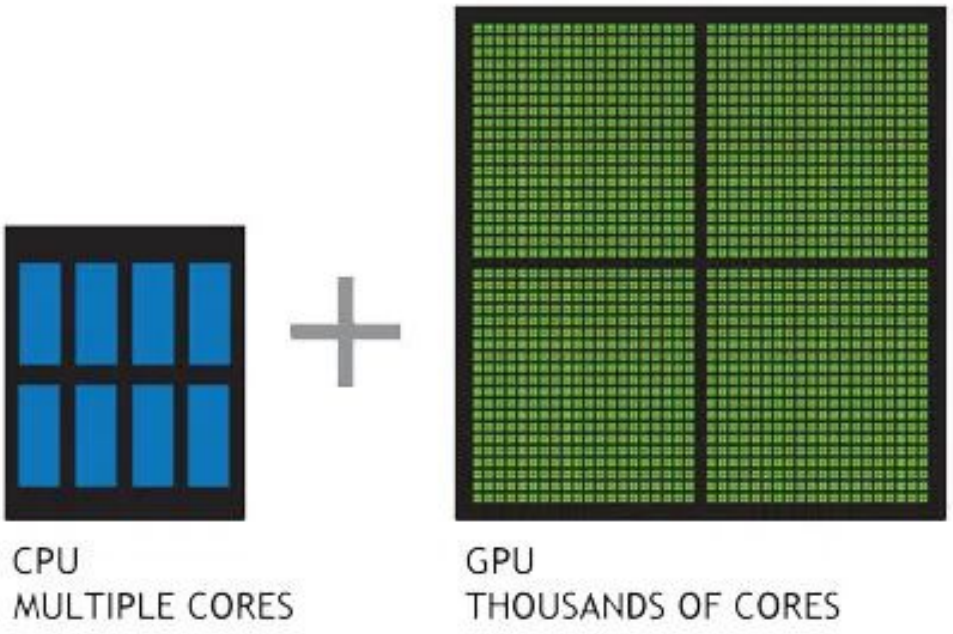
6.1.3. Ferramentas

A execução dos códigos de [10], com o dataset traduzido para português foi feita na plataforma *Google Colab*, uma plataforma online grátis do *Google* que provê um ambiente baseado em *Jupyter Notebooks* para execução de códigos em *Python* e *R*.

A grande vantagem na utilização desta plataforma para a tarefa deste trabalho se dá devido à possibilidade de utilização de infraestrutura do *Google* para execução dos códigos, sendo importante ressaltar a possibilidade de utilização de GPUs (*Graphic Processing Unit*). Este tipo de hardware é muito importante em tarefas relativas a processamento de linguagem natural pois, por ser especializado em operações com matrizes, o processamento pode ser paralelizado de forma que os tempos de execução são muito reduzidos quando comparados aos tempos obtidos para execução em CPUs (*Central Processing Unit*). Isto porque, simplificando, as CPUs possuem poucos núcleos de grande complexidade, sendo

sua arquitetura focada em otimizar a performance em *threads* únicas, enquanto que GPUs possuem muitos núcleos de menor complexidade, possibilitando milhares de *threads* simultâneas. Uma comparação ilustrativa pode ser vista na figura 6.6:

Figura 6.6: Ilustração que mostra, simplificadamente, a diferença CPUs e GPUs



Fonte: Ciência e Dados. Disponível em <http://www.cienciaedados.com/gpu-e-deep-learning/cpu-and-gpu/>. Acesso em: 03 dez. 2020.

Em sua versão gratuita, o *Google Colab* disponibiliza GPUs e, por isso, optou-se por utilizar essa plataforma pois além de prover o *hardware*, ela é independente do computador pelo qual está sendo acessado, sendo ideal para a colaboração no desenvolvimento. Um ponto relevante é que nesta versão, as especificações de GPUs disponibilizadas podem variar dependendo do momento de execução, podendo ter desempenho variável. Ainda assim, a diferença no tempo de execução dos testes é substancial comparando CPU e GPU. Com a primeira, os testes demoram horas e os treinamentos demoram dias para serem executados, enquanto que com a GPU, os testes passam a demorar alguns minutos e os treinamentos algumas horas. Um outro ponto importante é que o

Google Colab limita o tempo de execução ininterrupto a 12 horas o que, na fase de ajuste fino, é um limitador.

6.1.4. Implementação

Nesta primeira abordagem, os esquemas são traduzidos de português para inglês, de forma a serem resolvidos na língua inglesa. Como o desenvolvimento da resolução de desafios de esquemas de Winograd já é bastante avançado em inglês, não faria sentido construir uma nova solução para o desafio neste idioma.

Assim, optou-se pela busca de soluções satisfatórias que disponibilizassem a solução em código aberto, de modo a comparar os resultados originais em inglês com os resultados obtidos com os esquemas traduzidos de português para inglês. A solução com melhores resultados no desafio que disponibiliza a implementação em código aberto é a solução de [10]. É importante ressaltar que essa implementação já disponibiliza o BERT com ajuste fino no *dataset* WSCR (chamado de *Definite Pronoun Resolution dataset)*, não sendo necessário repetir este processo.

Esta solução, como já apresentado na seção 5.2, tem uma acurácia de de 61,9% com o BERT-Large original, sem ajuste fino. Com o ajuste fino sobre o BERT-Large com o *dataset* WSCR, chega-se a uma acurácia de 72,5%. À época da publicação, esse trabalho foi o estado da arte na solução do WSC, posteriormente ultrapassado por outros trabalhos. A implementação em código aberto é disponibilizada no *Github* do autor.

Nesta etapa, como desejamos comparar os resultados já citados, a partir dos esquemas do WSC originais em inglês, com os esquemas traduzidos de português para inglês, não é necessário fazer alterações no código, bastando alterar o dataset de testes conforme o teste a ser realizado.

6.1.4.1. Preparação do Ambiente no Colab

Após a definição da solução e metodologia a serem utilizadas, o passo inicial para possibilitar os testes é a criação e preparação do ambiente de execução de código no Colab. O ambiente criado é aberto para visualização para emails

vinculados à Universidade de São Paulo e tem título "*Winograd Schema Challenge in Portuguese*"[2].

A primeira dificuldade encontrada no trabalho no *Colab* foi em relação ao armazenamento de arquivos. Inicialmente, o repositório do *Github* de [10] era clonado no *Colab* todas as vezes em que se desejava executar a solução. Porém, esta abordagem se mostrou ineficiente pois qualquer alteração em algum arquivo ou na estrutura de arquivos não era persistida, tendo que ser repetida toda vez que se executava a solução. Além disso, o modelo com ajuste fino e o modelo em português são arquivos grandes que precisavam ser carregados em todas as execuções, o que é um processo longo e que se mostrou impraticável.

Tendo isto em vista, dado o acesso dos alunos da Universidade de São Paulo às soluções do *Google*, optou-se por usar o *Google Drive* como repositório compartilhado de arquivos, de modo que as mudanças e os novos arquivos adicionados fossem persistidos. Felizmente, o *Colab* possui uma biblioteca que conecta o sistema de *runtime* (tempo de execução) a um diretório do *Google Drive*, funcionando como se fossem arquivos locais do sistema de *runtime*. Vale ressaltar que, por segurança, todas as vezes que o sistema de *runtime* é reiniciado, é preciso autorizar a conexão ao *Google Drive* com a conta *Google* (vinculada à USP).

A próxima etapa é a instalação no ambiente das dependências do código como bibliotecas e pacotes. Essas dependências são especificadas no *notebook* do *Colab*[2]. Vale ressaltar que a instalação dessas dependências não é persistente, tendo que ser executadas a cada vez que o sistema de *runtime* é reiniciado. Porém, este processo não é demorado, o que viabiliza a execução a cada reinício.

Nesta primeira abordagem, como não são realizados treinamentos de ajuste fino, apenas a execução do dataset de teste é necessária. Os arquivos utilizados para teste ficam dentro da pasta *data*, do código de [10]. Diversos testes são executados, em diferentes *datasets*, porém, modificando o código, apenas o teste no WSC273 foi mantido. O arquivo com este teste é o *"wsc273.txt"*. Para possibilitar a execução dos modelos tanto no *dataset* original, como no *dataset*

2. Projeto de Formatura - Desafio de Esquemas de Winograd (WSC) em Português. Notebook do projeto no *Google Colab*. Disponível em <https://colab.research.google.com/drive/1i2GuYpxcvjJl7xSUavU8F2s42hDQn2kh?usp=sharing>. Acesso em: 30 nov. 2020.

traduzido, optou-se por manter ambos os *datasets* com o nome já citado, mas em pastas diferentes. O *dataset* original, em inglês, fica no caminho "*data/"* e o *dataset* traduzido fica no caminho "*data/portuguese/translation/"*. Para selecionar algum dos dois, basta mudar um dos parâmetros de execução do código, que será visto a seguir. O formato dos dados nesses arquivos são explicitados na seção 6.1.5.

Para a execução do código de [10], basta seguir as instruções do autor, usando o seguinte formato de comando:

```
python main.py \
      --task_name wscr \
      --do_eval \
      --eval_batch_size 10 \
      --data_dir "data/" \
      --bert_model bert-large-uncased \
      --max_seq_length 128 \
      --output_dir model_output/
```

O comando apresentado acima executa a tarefa "*wscr"*, que foi adaptada manualmente para executar apenas o teste no WSC273. A presença do parâmetro "*--do_eval*" indica que apenas a avaliação será realizada. O parâmetro "*--eval_batch_size*" é o tamanho do *batch* de avaliação e foi mantido sempre em 10 por padrão do autor. O parâmetro "*--data_dir"* é o comando citado anteriormente, que indica qual o diretório de dados que contém os arquivos utilizados no teste. Como já dito, para usar a versão original do WSC273 deve-se inserir "*data/"* e para usar a versão traduzida de português para inglês "*data/portuguese/translation/".* O parâmetro "*--bert_model"* indica qual o modelo do BERT a ser utilizado, estando disponíveis diversos modelos, incluindo *bert-large-uncased, bert-base-uncased e bert-base-multilingua*l. O BERT em português foi adicionado manualmente posteriormente, visto na seção 6.2.2. O parâmetro "*--max_seq_length*" indica o comprimento máximo das sequências lidas e foi mantido no valor padrão 128 para todas as execuções. O parâmetro *"--output_dir"* indica o diretório onde são salvos dados de saída.

Caso deseje se utilizar os modelos com ajuste fino disponibilizados para download pelo autor, basta acrescentar o parâmetro “*load_from_file”* com o caminho do arquivo do modelo, que deve ser salvo na pasta “*models*”. Por exemplo, para utilizar o modelo BERT_Wiki_WSCR que é o modelo que atingiu os melhores resultados, basta acrescentar a seguinte linha no comando de execução:

```
--load_from_file models/BERT_Wiki_WscR
```

Os comandos para treinamento de ajuste fino serão apresentados na seção 6.2.2, referente à segunda abordagem.

6.1.4.2. Entendimento da Solução em Código Aberto

Apesar da utilização de um ambiente em código aberto já pronto, foi muito importante durante o trabalho ter um entendimento claro dos comandos realizados e da lógica utilizada no código, de modo que fosse possível adaptá-lo às necessidades deste trabalho.

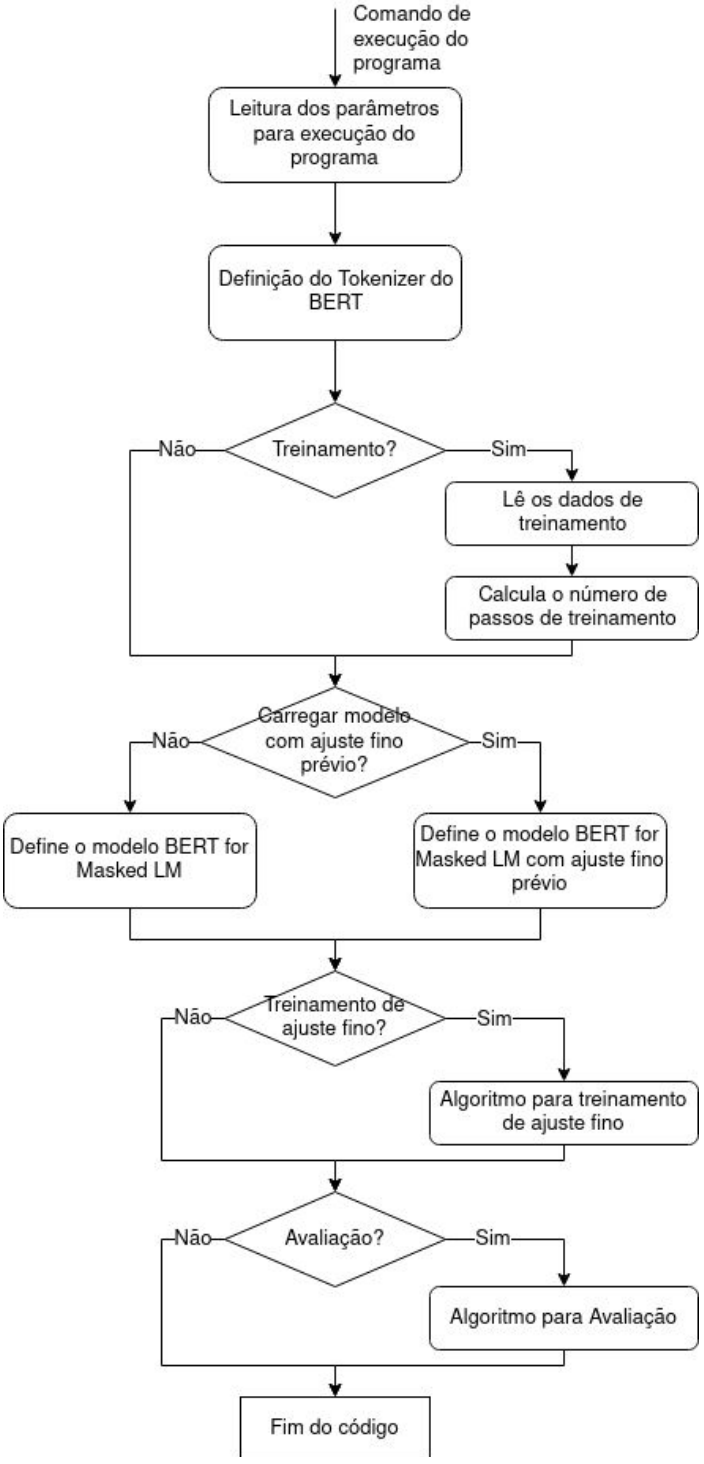
Para um entendimento em alto nível da estrutura do código, criou-se então diagramas que apresentam as etapas realizadas. O primeiro diagrama, visto na figura 6.7, apresenta as etapas realizadas de forma geral, entrando no detalhe da implementação do treinamento de ajuste fino e avaliação nas figuras 6.8 e 6.9, respectivamente.

Neste primeiro diagrama, na Figura 6.7, podemos ver que o primeiro passo do programa é ler os parâmetros de execução citados na subseção anterior. Em seguida, é definido qual *Tokenizer* será utilizado, sendo esta a lógica utilizada para transformar os dados de entradas em *tokens* (palavras inteiras ou subdivisões de palavras) compreensíveis pelo modelo por estarem presentes em um vocabulário pré-definido para tal modelo.

Em seguida, caso o parâmetro “--do_train” seja incluído nos parâmetros de execução, indicando a realização de treinamento de ajuste fino, os dados textuais que serão usados no treinamento são lidos e já é calculado o número de passos de treinamento, baseado nos parâmetros de entrada (os parâmetros usados para execução do treinamento de ajuste fino serão explicitados na seção 6.2.2).

O próximo passo é a definição do modelo utilizado, carregando ou não modelos com ajuste fino previamente realizado, salvos como arquivos. Por fim, baseado na presença dos parâmetros “--do_train” e “--do_eval” são executadas as etapas de treinamento para ajuste fino e de avaliação.

Figura 6.7: Diagrama geral do código utilizado



Fonte: Confecção própria.

Em relação ao trecho do código que executa a etapa de treinamento de ajuste fino, ilustrado na figura 6.8, a primeira etapa é a conversão dos exemplos presentes no *dataset* que será utilizado no treinamento para o formato adequado para ser lido pelo BERT. Este é o processo de *tokenização*, realizado através do *Tokenizer* definido anteriormente.

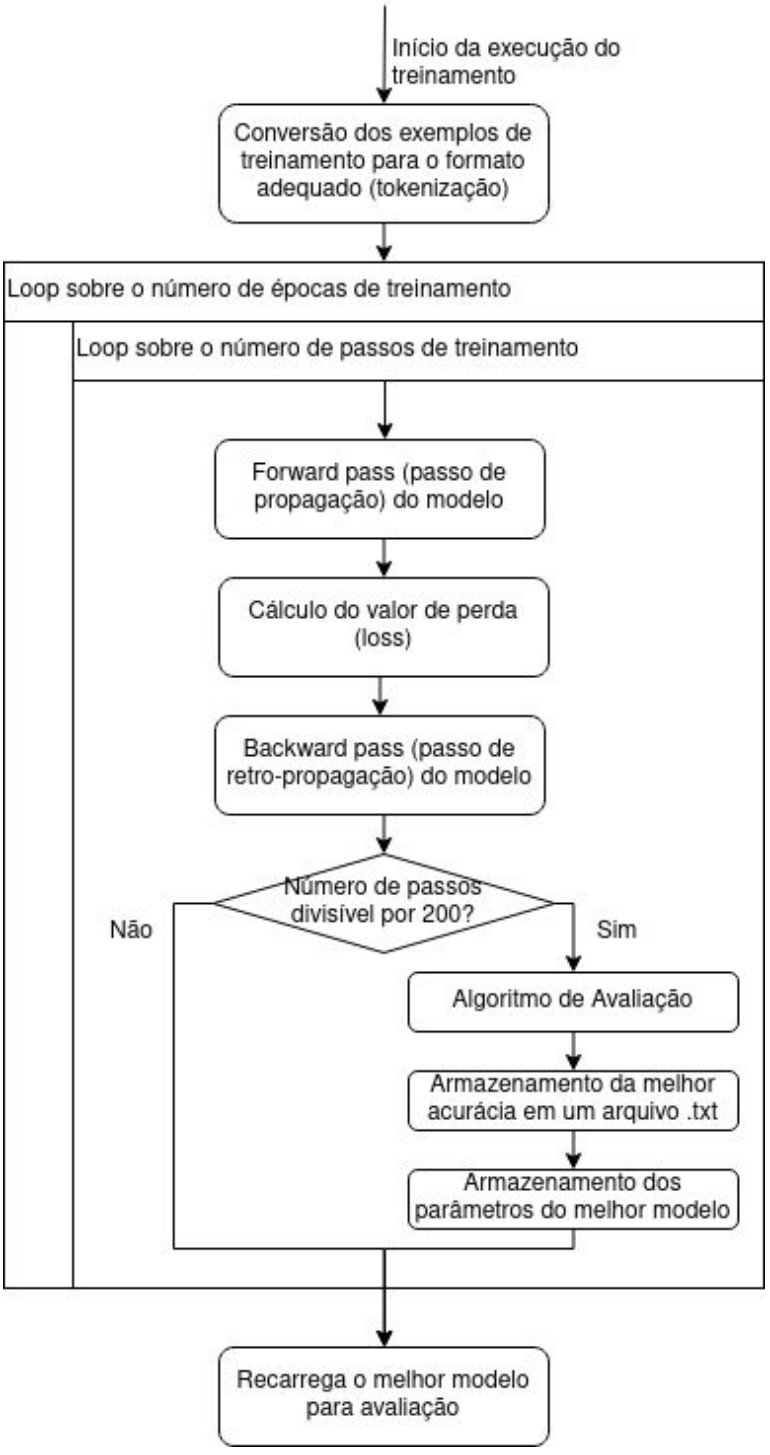
Em seguida, são definidos dois *loops* de execução, sendo o primeiro deles sobre o número de épocas de treinamento (*training epochs*), que é o número de vezes que o treinamento será repetido sobre o modelo. O *loop* interno a este itera sobre o número de passos de treinamento, calculado anteriormente.

Dentro desses *loops* ocorre o treinamento para o ajuste fino em si. O fluxo de treinamento é o padrão dos treinamentos de algoritmos de aprendizado supervisionado, sendo iniciado pelo *forward pass* (passo de propagação), onde os dados do *dataset* de treinamento são lidos e é calculado o resultado na tarefa sendo realizada. No caso do ajuste fino para resolução do WSC, frases de estrutura parecida ao desafio são usadas como entrada e neste passo o algoritmo faz a predição de qual é o candidato correto. Após esta passagem é calculada a função de perda (*loss function*), que quantifica os erros de predição do algoritmo. Tendo esta função, é então executado o *backward pass* (passo de retro-propagação), em que os parâmetros do modelo são atualizados de modo a tentar otimizar a função de perda, utilizando o gradiente descendente para o cálculo.

A cada 200 passos de treinamento, o algoritmo de avaliação é executado, calculando o desempenho do modelo até aquele ponto. Um arquivo de texto na pasta "*model_output"* armazena o valor da melhor acurácia, iniciada em zero, com a qual é comparada a acurácia calculada. Caso a acurácia calculada seja maior que a melhor acurácia armazenada, a acurácia calculada passa a ser a melhor acurácia, sendo este valor armazenado no arquivo e, além disso, o modelo atual (com os parâmetros atuais) é armazenado também na pasta citada, de modo que, ao fim do treinamento, tenha-se o modelo com os parâmetros que se saíram melhor na avaliação.

Por isso, ao fim dos *loops* de treinamento, este modelo armazenado é então carregado, de modo que possa ser utilizado para a avaliação que ocorrerá ao fim do treinamento (caso o parâmetro --do_eval seja incluído na chamada do programa).

Figura 6.8: Diagrama da etapa de treinamento de ajuste fino do código utilizado



Fonte: Confecção própria.

Na figura 6.9 pode-se visualizar um diagrama da etapa de avaliação do código, explicitando como é calculada a acurácia do modelo em um determinado *dataset* de avaliação.

A etapa de avaliação é iniciada pela leitura dos exemplos deste *dataset* de avaliação, que é chamado de *test set*. Assim como na etapa de treinamento para ajuste fino, os exemplos lidos passam pelo processo de *tokenização* com o *Tokenizer* definido anteriormente, adaptando os exemplos textuais para o formato adequado para leitura do modelo.

Em seguida ocorre o *forward pass* (passo de propagação) do modelo, executado em lotes (*batches,* com *eval_batch_size* sendo um dos parâmetros de entrada já citados), semelhante ao ocorrido na etapa de treinamento, no qual o modelo calcula qual o candidato de maior probabilidade para cada esquema. Neste caso, para definir qual o candidato de maior probabilidade, calcula-se um valor de perda (*loss*) para cada um dos dois candidatos do esquema, sendo que quanto menor este valor, maior a probabilidade que o modelo dá para que aquele seja o candidato correto. A formulação matemática deste cálculo é disponibilizada em [10]. Isso é feito para todos os esquemas.

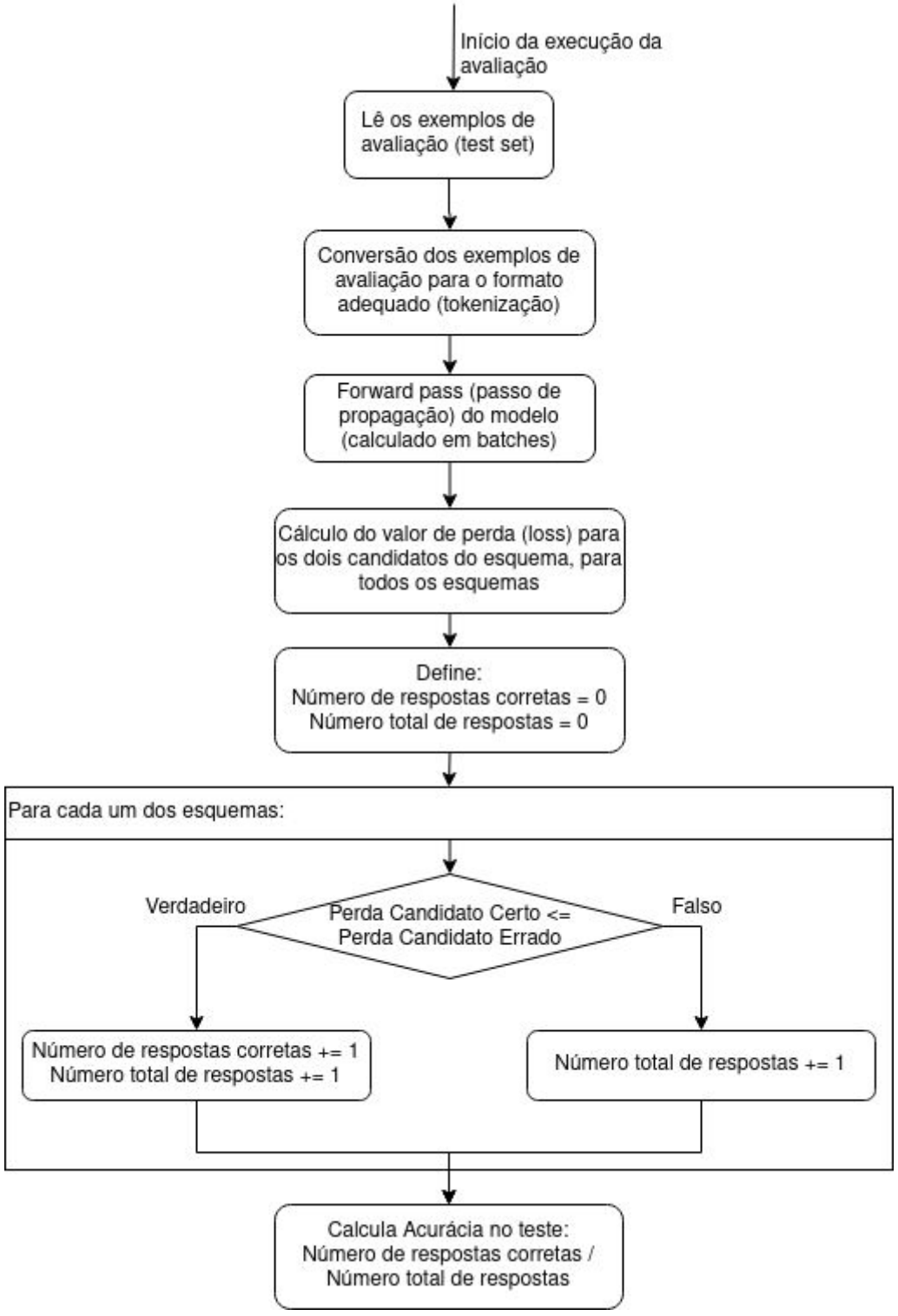
Tendo o valor de *loss* calculado para os dois candidatos de todos os esquemas, são então instanciadas duas variáveis, ambas com valor inicial zero, que serão utilizadas para o cálculo da acurácia:

- Número de respostas corretas
- Número total de respostas (corretas + incorretas)

Utiliza-se então um *loop* que itera sobre todos os esquemas. Dentro deste loop, compara-se a perda calculada para cada um dos candidatos. Como o *dataset* tem em seus dados qual é o candidato correto, se o valor da perda calculada para o candidato correto for menor ou igual à perda calculada para o candidato incorreto, diz-se que o modelo acertou a predição para aquele esquema. Neste caso, incrementa-se o valor de ambas as variáveis citadas anteriormente. No caso contrário, em que a perda calculada para o candidato correto é maior que a perda calculada para o candidato incorreto, diz-se que o modelo errou a predição do esquema e, então, apenas a variável “Número total de respostas” é incrementada.

Ao fim do *loop*, para calcular a acurácia basta dividir a variável com o número de respostas corretas pela variável com o número total de respostas, chegando ao valor desejado.

Figura 6.9: Diagrama da etapa de avaliação do código utilizado



Fonte: Confecção própria.

6.1.5. Avaliação dos resultados com tradução otimizada

Em [10], foi obtido uma acurácia de 61,9% com o BERT-Large original, sem modificações. Utilizando o *dataset* proveniente da tradução de português para inglês, com os ajustes supracitados, obteve-se uma acurácia de 61,4% Descontando os 12 esquemas adicionais dos 285 traduzidos e calculando a precisão no WSC273, a precisão com os esquemas traduzidos foi semelhante à precisão no *dataset* original, de 61,9%. Apesar de acertar a predição no mesmo número de esquemas, os esquemas nos quais houve acerto mudaram dependendo do *dataset.*

Porém, em [10] foram obtidos resultados ainda melhores utilizando versões do BERT com ajuste fino. Dentre todas as versões, o melhor resultado apresentado no artigo foi de 72,5% com a versão BERT_WIKI_WSCR que é uma versão do BERT pré-treinada em um *dataset* denominado *MaskedWiki* com ajuste fino no *dataset* WSCR. Utilizando a versão do WSC traduzido de português para inglês, com os ajustes já citados, a versão do BERT que teve melhores resultados também foi a BERT_WIKI_WSCR, chegando a um resultado de 72,9%. Este resultado ter sido superior ao resultado obtido com o modelo com ajuste fino no *dataset* original em inglês pode ser decorrência de diversos fatores, como por exemplo, erros na tradução que favoreceram a resposta certa, traduções em que o gênero indicou a resposta correta, dentre outros. Os resultados são apresentados na tabela 6.3.

Tabela 6.3: Comparativo de resultados entre modelos e *datasets* utilizados.

Modelo	Dataset	Acurácia obtida
BERT-Large original	WSC273 original	61,9%
BERT-Large original	WSC273 traduzido	61,9%
BERT_WIKI_WSCR	WSC273 original	72,5%
BERT_WIKI_WSCR	WSC273 traduzido	72,9%

A fim de destrinchar esse resultado, analisaram-se os problemas com a tradução do *dataset* de teste e com a adaptação do mesmo para utilização no

modelo, comparando as diferentes precisões para cada um dos casos, de modo a livrar-se de qualquer viés que poderia estar incluso. Sendo assim, os principais problemas encontrados com a tradução dos 273 esquemas foram:

- Pronome nos campos “pronoun” e “snippet” traduzidos de forma diferente do que o presente no campo “schema”. Exemplo:

{
"schema": "The delivery truck passed the school bus quickly because **it** was going too fast.",
"snippet": "**he** was going too fast.",
"pronoun": "**he**",
"correct_answer": "A",
"substitution_a": "The delivery truck",
"substitution_b": "The school bus",
"translated": true,
"pronoun_from_snippet": "he"
}

- Pronome no campo “pronoun” traduzido diferente do campo “schema”. Exemplo:

{
"schema": "The medal does not fit in the briefcase because **it** is too big.",
"snippet": "it is very big.",
"pronoun": "**she**",
"correct_answer": "A",
"substitution_a": "the medal",
"substitution_b": "to suitcase",
"translated": true,
"pronoun_from_snippet": "it"
}

- Pronome no campo “snippet” traduzido diferente do campo “schema”. Exemplo:

{
"schema": "George got free tickets for the play, but he gave them to Eric, even though **he** was particularly interested in watching it.",
"snippet": "**him** being particularly interested",
"pronoun": "he",
"correct_answer": "A",
"substitution_a": "George",
"substitution_b": "Eric",
"translated": true,
"pronoun_from_snippet": "him"
}

- Casos em que um apóstrofo foi inserido erroneamente na tradução. Exemplo:

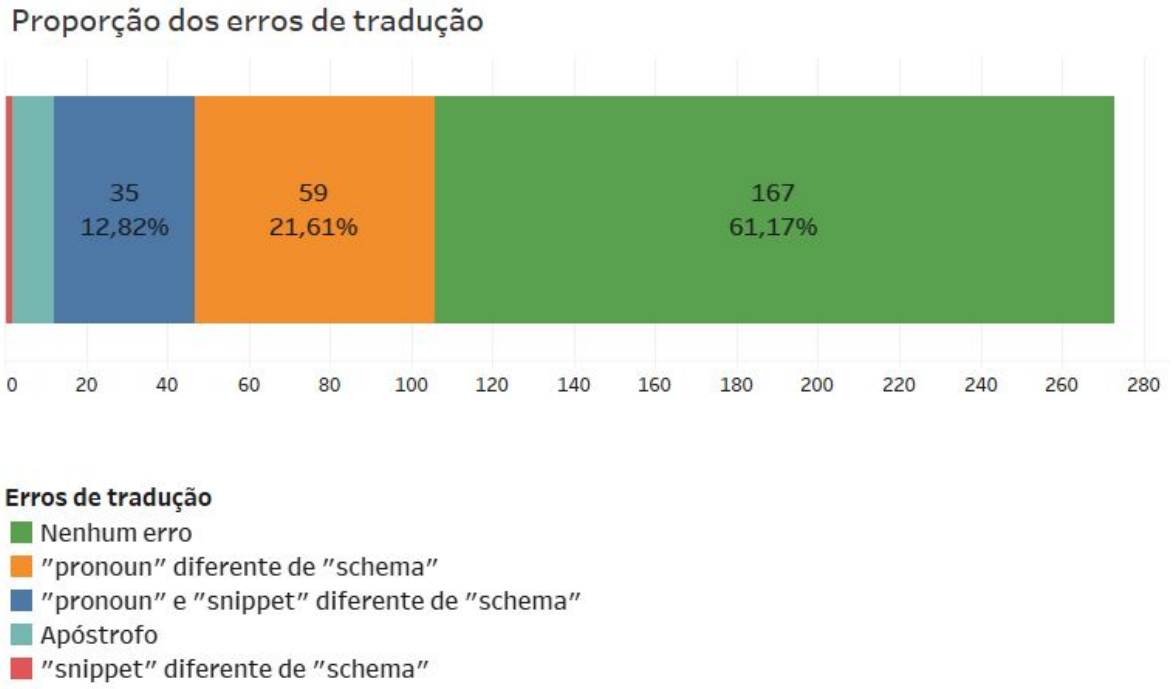
{
"schema": "John couldn't see the stage with Billy in front of him because **he is** too tall.",
"snippet": "**He's** too tall.",
"pronoun": "he",
"correct_answer": "B",
"substitution_a": "John",
"substitution_b": "Billy",
"translated": true,
"pronoun_from_snippet": ""
}

Tendo em vista tais erros com a tradução, fez-se necessário analisar a proporção entre eles, e quais os impactos de cada um no modelo. De acordo com a figura 6.10 abaixo, pode-se perceber que em 61% dos casos, o equivalente a 167

esquemas, não houve nenhum erro de tradução. Já em pouco mais de 21% dos casos, o campo “pronoun” teve tradução diferente do pronome no campo “schema”, o que revelou-se um empecilho na etapa de mascarar os pronomes, mas que pôde ser resolvido na própria etapa de adaptação do *dataset* de teste. Em 12,8% dos casos tanto o campo “pronoun” quanto “snippet” tiveram traduções diferentes do pronome no “schema”, o que gerou o mesmo desafio já descrito, com relação ao campo “pronoun”, somado ao fato de que o campo “snippet” é ainda mais confiável que o “pronoun” por possuir o contexto da ambiguidade, então ele ser traduzido erroneamente dificulta também que o pronome ambíguo seja mascarado corretamente, mas, novamente, esse erro pode ser resolvido na fase de adaptação do *dataset.*

Por fim, os casos em que o “snippet” foi traduzido diferente da real ambiguidade do “schema” são realmente prejudiciais, e tratados mais a frente, enquanto que os casos com adição de apóstrofo incorretamente não foram prejudiciais para o *dataset* de entrada do modelo, não resultando em tratamentos específicos.

Figura 6.10: Proporção dos erros de tradução.



Fonte: Confecção própria.

Após o processo de tradução, foi necessário adaptar o *dataset* para poder utilizar o modelo, substituindo o pronome ambíguo por uma máscara [MASK], requisito para funcionamento adequado da tarefa de *Masked Language Modeling* do BERT, onde o modelo prevê quais palavras tem maior probabilidade de serem a substituta correta do [MASK]. O formato adaptado a ser armazenado no arquivo de texto é composto em sua primeira linha pelo esquema com o pronome substituído por [MASK], pelo texto [MASK] na segunda linha, seguido pelos dois substantivos candidatos na terceira linha, separados por vírgula, estando o candidato correto na quarta linha.

Nesta etapa, o campo “pronoun” era usado para fazer a substituição do pronome no campo “schema” por [MASK], ou seja, o *script* checava quando o valor do campo “pronoun” era igual a alguma palavra do campo “schema” e substituia o [MASK] no caso dessa igualdade ser verdadeira.

Porém, ao utilizar essa lógica de substituição, nos casos em que o campo “pronoun” era traduzido de forma diferente do pronome no campo “schema”, o algoritmo não encontrava a igualdade e, portanto, não fazia a substituição do [MASK] corretamente. Um caso como esse pode ser visto abaixo:

- Tradução do esquema para o inglês:

  {

  "schema": "The medal does not fit in the briefcase because it is too big.",

  "snippet": "it is very big.",

  "pronoun": "she",

  "correct_answer": "A",

  "substitution_a": "the medal",

  "substitution_b": "to suitcase",

  "translated": true

  }

- Estado após rodar o algoritmo inicial de substituição por [MASK], adaptando para o formato citado anteriormente:

  The medal does not fit in the briefcase because it is too big.

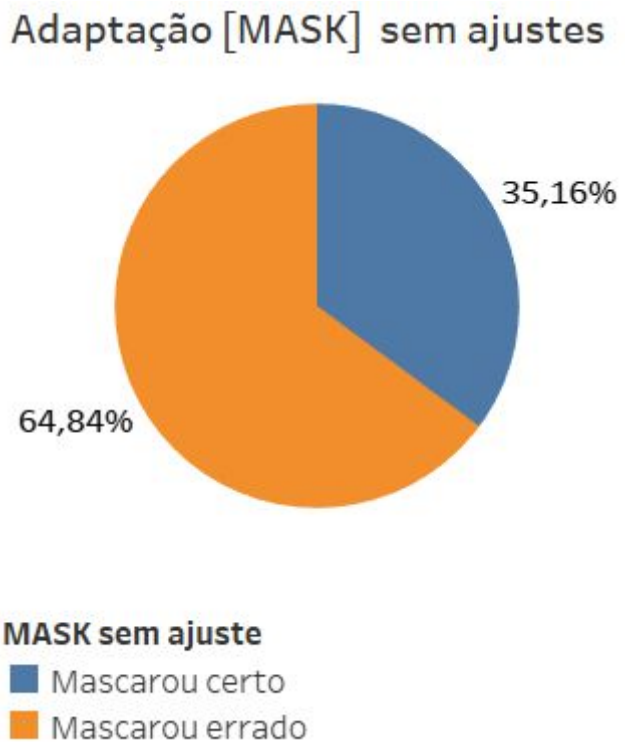
[MASK]

the medal, to suitcase

the medal

Com isso, observaram-se diversos casos em que a substituição da máscara não se deu corretamente, mais precisamente em aproximadamente 64,84% dos 273 esquemas traduzidos e adaptados, como representado na figura 6.11 abaixo. Isso prejudicaria consideravelmente o desempenho do modelo, por utilizar um *dataset* com mais da metade dos dados de treinamento fora do padrão de mascaramento.

Figura 6.11: Proporção entre esquemas mascarados corretamente e erroneamente, sem ajustes.



Fonte: Confecção própria.

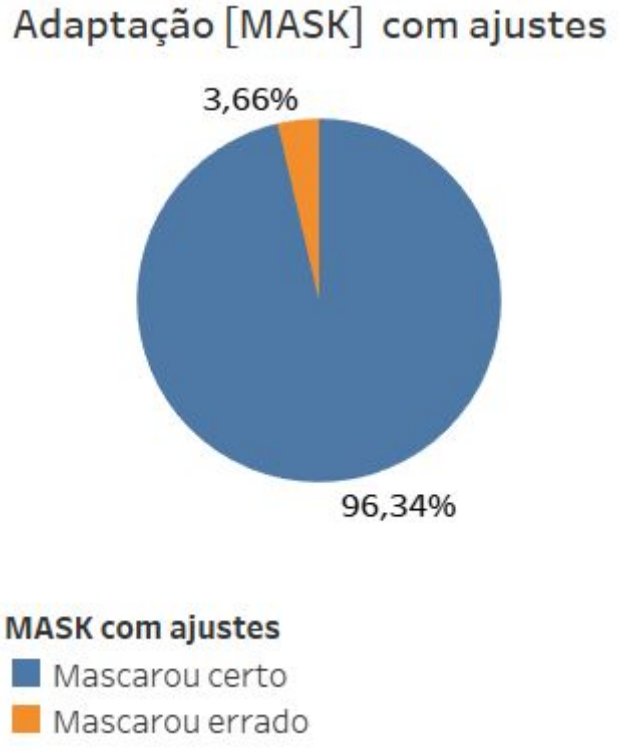
A fim de ajustar esses erros de substituição do MASK, complementou-se o *script* de adaptação com algumas lógicas:

- Criou-se uma lista com os possíveis pronomes utilizados em estruturas gramaticais como a definida pelo WSC:
    - pronouns = *["he", "her", "him", "his", "it", "she", "them", "they"]*

- Sempre que o tradutor faz a tradução de um pronome sem o contexto, ou seja, no campo “*pronoun*” que só possui o pronome, sem demais palavras, criou-se uma lógica para verificar se o campo está correto. Por exemplo, para o pronome “*your*” que não é válido para o desafio, procura-se os pronomes “*his*” ou “*her*” no esquema, e, ao encontrá-lo, substitui-se no campo.
- Caso o pronome do campo “*snippet*” não seja encontrado no esquema, uma lógica percorre o esquema de trás para frente - dado que o pronome ambíguo se encontra no final das frases em sua maior parte - pegando o primeiro pronome que aparece na frase e existe na lista de pronomes possíveis, substituindo esse pronome no campo “*snippet*”.

Após os ajustes descritos acima, conseguiu-se reverter o cenário de adaptação, chegando a 96,34% de acerto ao mascarar os pronomes como mostrado na figura 6.12 abaixo. Vale ressaltar que após essas correções, todos os esquemas são considerados válidos, pois possuem o campo [MASK], mesmo que na posição errada.

Figura 6.12: Proporção entre esquemas mascarados corretamente e erroneamente, com ajustes.



Fonte: Confecção própria.

Nesse processo, mesmo após os ajustes feitos, cerca de 3,5% dos esquemas tiveram o [MASK] substituído no pronome errado. Ao analisar esses 10 casos, notou-se:

- Em 2 dos 10 casos o pronome do campo “snippet” não foi encontrado no campo “schema” e o algoritmo, nessa condição, busca pelo último pronome da lista de pronomes para mascarar. Entretanto, o que pode acontecer é de o primeiro pronome de trás para frente não ser o ambíguo, o que acarreta a substituição errada do [MASK]. Exemplo:
  - Esquema traduzido para o inglês

    {

    "schema": "George got free tickets for the play, but he gave them to Eric, even though **he** was particularly interested in watching **it**.",

    "snippet": "him being particularly interested",

    "pronoun": "he",

    "correct_answer": "A",

    "substitution_a": "George",

    "substitution_b": "Eric",

    "translated": true,

    "pronoun_from_snippet": "him"

    }
  - Esquema mascarado após o algoritmo

    George got free tickets for the play, but he gave them to Eric, even though **he** was particularly interested in watching [MASK].

    [MASK]

    George, Eric

    George
- Nos outros 8 casos, o pronome no campo “snippet” é encontrado no esquema, porém existem dois pronomes iguais na mesma frase,

sendo o ambíguo o primeiro, não o último como o algoritmo prevê que seja. Exemplo:

- Esquema traduzido para o inglês

{

"schema": "I saw Jim yelling at a guy who wore a uniform and had a big red beard. I don't know why **he** was like this, but **he** looked very upset.",

"snippet": "because **he** was like this",

"pronoun": "he",

"correct_answer": "A",

"substitution_a": "Jim",

"substitution_b": "The guy who wore a uniform",

"translated": true,

"pronoun_from_snippet": "he"

}

- Esquema mascarado após o algoritmo

I saw Jim yelling at a guy who wore a uniform and had a big red beard. I don't know why he was like this, but [MASK] looked very upset.

[MASK]

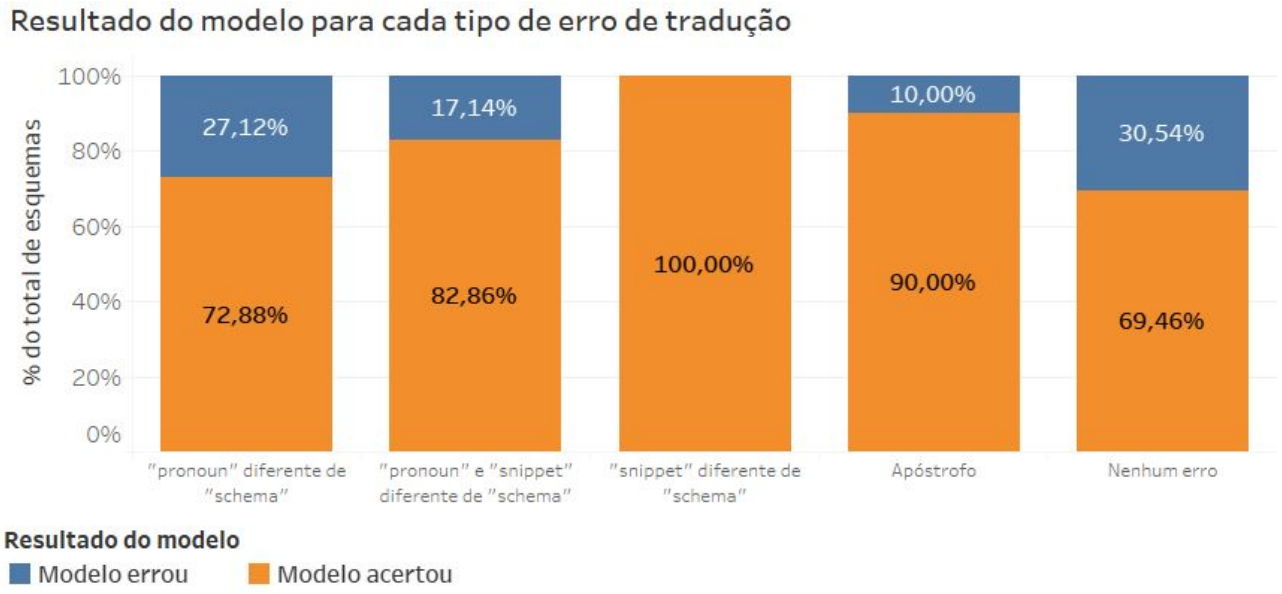
Jim, The guy who wore a uniform

Jim

Como os casos que representam estes 3,5% de mascaramentos errôneos não são prejudiciais para a compreensão do esquema, ou seja, não acarretam na perda do sentido da frase, e possuem a máscara, mas não no pronome ambíguo, concluiu-se que não seria necessário desenvolver novos algoritmos que minimizariam esses erros, dado que os ganhos não seriam significativos. Ainda assim, é importante comparar os resultados da execução do modelo BERT_WIKI_WSCR ao descontar tais casos da precisão total, como será apresentado à frente.

O gráfico apresentado a seguir, na figura 6.13, representa a porcentagem de acertos e erros do modelo para cada caso dos problemas de tradução, e para o caso em que não houve nenhum erro. Aqui ainda não são descontados os casos em que [MASK] foi substituído erroneamente, somente são mostrados os erros de tradução e como se manifestaram no modelo.

Figura 6.13: Proporção de Acertos e Erros do modelo para cada caso dos problemas de tradução.



Fonte: Confecção própria.

Pode-se observar que o modelo errou mais, proporcionalmente, nos casos em que não houve nenhum erro de tradução, do que quando tiveram erros como “pronoun” diferente do campo “schema”. Isto faz sentido, visto que o *dataset* que entra no modelo é o mascarado, cujo campo “pronoun”, por exemplo, não é utilizado pelo algoritmo de substituição de [MASK]. Logo, os erros que podem ser mais prejudiciais para o desempenho dos modelos em geral, não só o BERT_WIKI_WSCR, são aqueles em que o [MASK] foi substituído no lugar errado (em que mesmo com a tradução correta, a substituição foi feita em outro pronome da frase que não o ambíguo), por ser o dataset que é efetivamente a entrada do modelo.

A fim de comprovar tal afirmação, através da Tabela 6.4 abaixo, filtrou-se somente os casos em que o [MASK] foi substituído erroneamente. Pode-se observar que a proporção de erros do modelo para cada um dos casos, inclusive

os casos que a tradução estava correta, aumentou de aproximadamente 30% para 50%. Com isso, conclui-se que, de fato, essa substituição errônea prejudica o desempenho do modelo e, portanto, pode-se desconsiderar os casos em que [MASK] foi substituído errado para fins de cálculos da acurácia final.

Tabela 6.4: Acurácia para os casos em que a máscara foi substituída corretamente.

Acurácia para cada caso de erro de tradução, com MASK errado

Erros de tradução	Acurácia	Resultado	Contagem de Esquemas
"pronoun" diferente de "schema"	50,00%	2	4
"pronoun" e "snippet" diferente de "schema"	50,00%	1	2
"snippet" diferente de "schema"	100,00%	2	2
Nenhum erro	50,00%	1	2

Descontando os casos em que [MASK] não foi substituído corretamente, obtiveram-se as seguintes acurácias para cada caso de tradução, representadas na Tabela 6.5:

Tabela 6.5: Precisão de cada caso de tradução, descontando-se casos com [MASK] errado.

Acurácia para cada caso de erro de tradução, com MASK correto

Erros de tradução	Acurácia	Resultado	Contagem de Esquemas
"pronoun" diferente de "schema"	74,55%	41	55
"pronoun" e "snippet" diferente de "schema"	84,85%	28	33
Apóstrofo	90,00%	9	10
Nenhum erro	69,70%	115	165

De posse dos conhecimentos do impacto de cada problema de tradução e dos casos com [MASK] errado na precisão total, pode-se concluir alguns pontos. Primeiro, a precisão levando em consideração todos os esquemas traduzidos foi de 72,9% utilizando o BERT_WIKI_WSCR, um pouco superior que a obtida utilizando o WSC original em inglês, o que é bastante satisfatório.

A fim de analisar melhor os impactos de cada erro nos resultados, é possível destrinchar as acurácias em outras três, além da original (72,9%) que considera todos os erros e acertos. A primeira obtém-se quando desconsidera-se do cálculo de acurácia os casos em que [MASK] foi substituído erroneamente, mesmo que esses casos não causem perda de sentido da frase, obtendo-se uma precisão maior, de 73,4%.

A segunda acurácia desconsidera os erros de tradução que impactam o *script* que realiza a substituição de [MASK]. Visto que o *script* que realiza a substituição de [MASK] no *dataset* é feito baseando-se no campo “snippet”, casos de problemas na tradução em que somente o campo “pronoun” foi traduzido de forma diferente do campo “schema” não influenciam na precisão do modelo. Desta forma, este tipo de erro não precisa ser desconsiderado em um cálculo desta precisão mais crítica, logo, esta desconsidera: casos em que [MASK] foi substituído errado, casos em que o “pronoun” e “snippet” foram traduzidos diferente do campo “schema”, erros de apóstrofo e erros em que o campo “snippet” - este, sim, utilizado pelo *script* de substituição - foi traduzido diferente do “schema”. Com isto, obteve-se uma acurácia de 70,9%.

Sendo ainda mais críticos com relação aos resultados, chega-se a terceira acurácia possível, em que considera-se somente casos em que não houve absolutamente nenhum erro na tradução - intitulados como “Nenhum erro” na tabela 6.5 - e que a substituição de [MASK] foi feita corretamente. Para tanto, obtém-se uma precisão de 69,7%, que, apesar de ser a menor de todas, ainda é bastante satisfatória.

Após essa análise, percebeu-se que os valores são bastante próximos, mesmo com os descontos dos casos em que houve erro na tradução. Porém, após as correções feitas no algoritmo de mascaramento, citadas anteriormente, constatou-se que mesmo com esses problemas, todos os esquemas são válidos, ou seja, possuem [MASK] na frase, mesmo que na posição errada. Assim, optou-se por considerar como aceito o valor inicial obtido, de 72,9%, pois caso essa abordagem fosse usada para outros esquemas, de outro problema, esse seria o resultado obtido sem a supervisão e descontos manuais. Ainda assim, mesmo que fossem considerados os valores que desconsideram os casos com erros (73,4%,

70,9% e 69,7%), os valores obtidos ainda são satisfatórios, se aproximando bastante do valor obtido com ajuste fino na avaliação feita no *dataset* original em inglês.

6.2. Abordagem 2: Execução do Modelo de Linguagem em Português com ajuste fino no WINOGRANDE traduzido

A segunda abordagem possível para solução do desafio é a de utilizar um modelo pré-treinado em português ou treinar um modelo em português e fazer os testes com o *dataset* de [2] em português. Essa solução foi escolhida para ser realizada posteriormente por representar um maior grau de dificuldade, dado que não são conhecidos modelos em português com um bom desempenho nesse desafio.

Uma das opções citadas seria realizar a etapa de pré-treinamento do BERT em um corpus em português de grandes proporções, como a *Wikipedia* em português. Porém, esta opção foge ao escopo deste trabalho, pois demandaria um imenso poder computacional e dias de execução do pré-treinamento.

Ainda assim, dois modelos podem ser utilizados para este desafio: o modelo BERT da *Neuralmind*, proposto em [8], que é uma versão do BERT com pré-treinamento realizado em um corpus em português, e o BERT Multilingual, proposto em [7], uma versão multi-idiomas do BERT, com pré-treinamento em diversos idiomas, que possui compatibilidade com a língua portuguesa.

Em ambos os casos, a ideia é fazer o ajuste fino utilizando o *dataset* WINOGRANDE, que possui 44 mil esquemas com estrutura similar ao WSC, disponibilizado em [5]. Porém, este *dataset* foi originalmente publicado em inglês. Com isso, o primeiro passo é traduzir esses esquemas para português, para então poder utilizá-los no ajuste fino dos modelos.

6.2.1 Tradução do WINOGRANDE para o português

Para poder realizar o ajuste fino e testar os modelos anteriormente citados, o primeiro passo é a obtenção do *corpus* para ajuste fino em português. O arquivo de teste, com as ambiguidades na língua portuguesa, foi retirado do trabalho em [2].

Para tanto, foi preciso transformá-lo em um arquivo .json, mais fácil de ser trabalhado em Python do que o HTML, como explicado na subseção 6.1.1. Já em relação ao arquivo a ser utilizado para o ajuste fino dos modelos, optou-se por utilizar o *dataset* WINOGRANDE, de [5], composto por 44 mil esquemas, por possuir mais exemplos de esquemas no formato do Desafio de Esquemas de Winograd, o que contribui para uma melhor acurácia do modelo após o processo de ajuste fino e pela elevada acurácia obtida no WSC quando usado para ajuste fino de uma variação do BERT, visto em [5].

Entretanto, esses 44 mil esquemas foram disponibilizados em um formato conhecido como .jsonl (JSON Lines). Com isso, o primeiro passo foi a construção de um *script,* disponível no repositório do projeto[1], usando *Python* que convertesse o formato .jsonl para .json, utilizando a biblioteca *jsonlines.* Em seguida, alterou-se a função de tradução utilizada na primeira abordagem para que a linguagem objetivo da tradução se tornasse o português, não o inglês.

Outro ponto importante de ressaltar é o formato e os campos fornecidos em cada arquivo .jsonl, pois são diferentes dos campos da primeira abordagem. O primeiro campo fornecido é o “qID”, que é o ID do esquema. Em seguida, aparece o campo “*sentence*” com o respectivo esquema, além de “*option1*”, “*option2*” e “*answer*”, cujo conteúdo são as opções de resposta e a resposta certa (indicada numericamente), respectivamente. Um exemplo desses campos pode ser visto abaixo:

{
"qID": "3QHITW7OYO7Q6B6ISU2UMJB84ZLAQE-2",
"sentence": "Ian volunteered to eat Dennis's menudo after already having a bowl because _ despised eating intestine.",
"option1": "Ian",
"option2": "Dennis",
"answer": "2"
}

Uma das grandes diferenças entre esse formato e os trabalhados anteriormente é que nesse caso o pronome ambíguo é ocultado, aparecendo “_” no lugar do mesmo. Essa diferença causou impacto no momento da tradução, porque na função utilizada na primeira abordagem, usando os arquivos que possuem o pronome ambíguo, a lógica de tradução levava em conta a posição desses pronomes. Já na segunda abordagem usando esse novo formato, o que aconteceu em alguns casos foi que a tradução para o português, do *Google Cloud*, ignorou o símbolo “_” e retirou ele do esquema.

Com isso, buscou-se formas de fazer a tradução sem que a posição do pronome, representada pelo símbolo “_”, sumisse do esquema e encontrou-se uma solução bastante simples. Tal solução foi substituir, antes de traduzir para o português, o símbolo “_” para “[_]”. Isso fez com que a função de tradução do *Google Cloud* compreendesse que este era um símbolo que não poderia ser cortado e mantivesse o mesmo intacto no esquema.

A tradução em si foi feita em *batches,* ou seja, cada arquivo que foi fornecido foi traduzido em separado para verificar as traduções. Os arquivos contém quantidades de esquemas diferentes, que vão aumentando progressivamente. O arquivo *train_xs.jsonl* possui 160 esquemas, o *train_s.jsonl* possui 640 esquemas, *train_m.jsonl* possui mais de 2 mil esquemas, o *train_l.jsonl* possui mais de 10 mil esquemas e, por fim, o completo *train_xl.jsonl* possui mais de 40 mil esquemas.

Como são muitos esquemas e levaria um tempo considerável para traduzi-los sequencialmente, optou-se por utilizar *multiprocessing* do *Python* paralelizando a tarefa de tradução e reduzindo bastante seu tempo de execução.

A fim de analisar a qualidade da tradução realizada escolheu-se uma amostra dos 44 mil esquemas, mais especificamente, o arquivo *train_xs.jsonl* traduzido, com 160 esquemas, possibilitando a análise sentença a sentença, compilando os erros e seus tipos. Vale ressaltar que, neste caso, como não temos frases de referência em português que indiquem a tradução correta, não é possível aplicar o *score* de BLEU para avaliar a tradução. Encontrou-se quatro principais tipos de erros:

- Diferentes palavras do inglês, porém com mesmo significado em português, resultaram na mesma tradução para ambas:

- Exemplo do esquema em inglês:

  {

  "sentence": "He never comes to my **home**, but I always go to his **house** because the _ is smaller.",

  "option1": "home",

  "option2": "house",

  "answer": "1"

  }

- Respectiva tradução para o português, no formato adequado:

  Ele nunca vem na minha casa, mas eu sempre vou na casa dele porque o [MASK] é menor.

  [MASK]

  casa, casa

  casa

- No caso exemplificado, as diferentes palavras do inglês “*house*” e “*home*” tiveram a mesma tradução para o português “casa”.

- Artigo definido traduzido com gênero errado:
  - Exemplo do esquema em inglês:

    {

    "sentence": "The couple drank wine out of glasses instead of mugs, because the _ were dirty.",

    "option1": "mugs",

    "option2": "glasses",

    "answer": "1"

    }

  - Respectiva tradução para o português, no formato adequado:

    O casal bebeu vinho em copos em vez de canecas, porque **os** [MASK] estavam sujos.

    [MASK]

    canecas, óculos

    canecas

- Nesse caso, “the” foi traduzido para “os” mesmo sendo “canecas” a resposta correta. A frase correta em português se torna errada gramaticalmente.

- Tradução de número errada:
  - Exemplo do esquema em inglês:

    {

    "sentence": "To overcome behavioural bias, we need to focus more on changing the conscious actions instead of the unconscious actions because the _ actions are voluntary.",

    "option1": "unconscious",

    "option2": "conscious",

    "answer": "2"

    }
  - Respectiva tradução para o português, no formato adequado:

    Para superar o preconceito comportamental, precisamos nos concentrar mais na mudança das ações conscientes em vez das ações inconscientes, porque as ações [MASK] são voluntárias.

    [MASK]

    inconsciente, consciente

    consciente
  - Nesse caso, a tradução das opções deveriam ser no plural, sendo “conscientes” e “inconscientes”, dado o contexto.

- Nomes próprios traduzidos para outras palavras:
  - Exemplo do esquema em inglês:

    {

    “sentence”: "When buying a home, Patricia doesn't have as much money to spend as Tanya, so _ buys a 5 bedroom house.",

    "option1": "Patricia",

    "option2": "Tanya",

    "answer": "2"

}

- Respectiva tradução para o português, no formato adequado:

  Ao comprar uma casa, Patricia não tem tanto dinheiro para gastar quanto Tanya, então [MASK] compra uma casa de 1 quarto.

  [MASK]

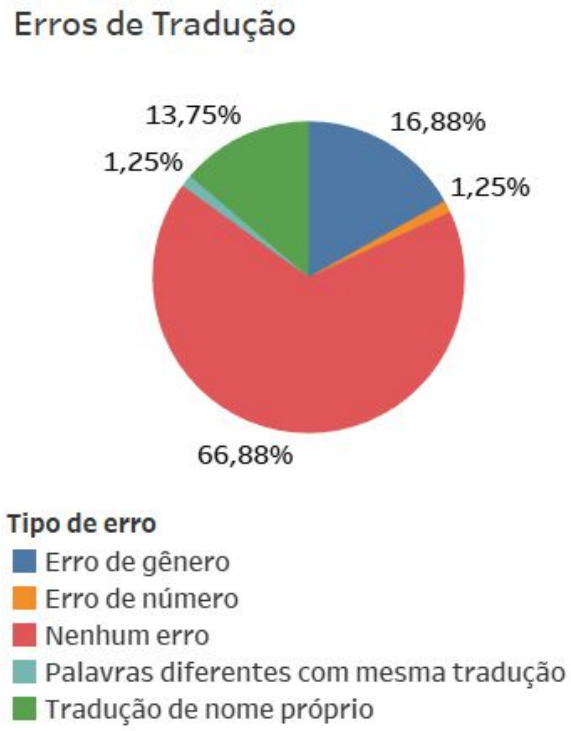
  Patricia, **Pergunte**

  Patricia

- Nesse caso, o nome “Tanya” foi traduzido para “Pergunte”, quando não deveria ser traduzido.

A proporção dos erros de tradução exemplificados acima, na base de 160 esquemas, pode ser visualizada pela figura 6.14 abaixo:

Figura 6.14: Proporção dos erros nos 160 esquemas.



Fonte: Confecção própria.

Pode-se observar que em quase 67% dos casos não encontrou-se nenhum tipo de erro, enquanto que os erros de número e gênero, que representam pouco

mais de 18%, não são erros que trazem grandes prejuízos para o treinamento, podendo considerar-se que 85% do *dataset* obteve uma tradução de qualidade.

Entretanto, os erros de tradução de nomes próprios, que representam quase 15% do dataset, são erros mais prejudiciais por não terem correspondência entre as opções e a sentença.

Porém, dado o alto volume de exemplos de treinamento e com a prevalência de 85% dos esquemas sem erros ou com erros aceitáveis, concluiu-se que a correção destes erros não era essencial para o ajuste fino dos modelos, pois apenas 15% dos esquemas seriam afetados. Além disso, neste caso, como os esquemas serão usados para o ajuste fino, o modelo é muito menos sensível a erros do que no caso de erros no *dataset* de avaliação, visto na seção 6.1. Deste modo, o arquivo *train_xl.jsonl,* com os 44 mil esquemas, sendo este o WINOGRANDE completo, foi traduzido e convertido para o formato adequado sem tratamentos adicionais, sendo então utilizado no ajuste fino dos modelos.

6.2.2. Implementação

Assim como na abordagem 1, utilizou-se para a execução e treinamento de ajuste fino dos modelos o código disponibilizado por [10], tendo sido feitas algumas alterações para adaptação do código para os modelos utilizados.

O modelo BERT Multilingual já estava incluído no código citado, bastando utilizá-lo como um dos parâmetros de execução do código. Já os modelos BERT Portuguese (Base e Large), disponibilizados pela *NeuralMind* em [8], tiveram de ser baixados e adicionados ao código manualmente.

Estes modelos estão disponíveis para *download* no Github da *NeuralMind*, em [8]. São disponibilizadas implementações através do *TensorFlow* e do *PyTorch*. Como o código utilizado usa a biblioteca *PyTorch*, optou-se por essa implementação. Foi realizado o *download* dos modelos e de seus respectivos vocabulários (em arquivo de texto), armazenando-os na pasta “portuguese”. Dentro desta, os arquivos do modelo Base foram armazenados na pasta “bert-base-portuguese-cased” e do modelo Large na pasta “bert-large-portuguese-cased”.

A primeira modificação necessária no código para que estes modelos em português funcionem adequadamente é a definição do *Tokenizer* correto, baseado no arquivo de vocabulário. Segundo o repositório do modelo em português, deve-se inserir o seguinte comando para definição do *Tokenizer:*

```
tokenizer = BertTokenizer.from_pretrained('path/to/vocab.txt', do_lower_case=False)
```

O argumento ‘path/to/vocab.txt’ deve ser substituído pelo caminho do arquivo de texto do vocabulário citado, sendo substituído por 'portuguese/bert-base-portuguese-cased/vocab.txt' para o modelo Base e 'portuguese/bert-large-portuguese-cased/vocab.txt' para o modelo Large. O código foi então alterado para que, caso o parâmetro “--bert_model” seja "bert-base-portuguese" ou "bert-large-portuguese", o *Tokenizer* é definido adequadamente, como citado. Caso contrário, o *Tokenizer* é definido da forma padrão para outros modelos (que já estava presente no código).

Seguido da adição do *Tokenizer* foram também inseridos os trechos de código citados no repositório do BERT Portuguese para incluir estes modelos. A lógica de inserção foi semelhante à do *Tokenizer*, criando condições para o caso dos parâmetros se referirem aos modelos em português e mantendo o código original no caso contrário. O comando para inserção dos modelos é no seguinte formato:

```
model = BertModel.from_pretrained('path/to/bert_dir', cache_dir = PYTORCH_PRETRAINED_BERT_CACHE / 'distributed_{}'.format(args.local_rank))
```

Assim como no *Tokenizer*, ‘path/to/bert_dir’ é substituído pelos respectivos caminhos onde os modelos foram salvos. O segundo argumento é padrão e foi mantido como na definição dos outros modelos.

Uma outra modificação realizada foi a adição de um novo nome de tarefa de treinamento, denotada como “portuguese-winogrande”, de modo que ao passar este campo como o parâmetro “--task_name”, o código executado realiza o

treinamento no *dataset* do WINOGRANDE traduzido. Para que isso fosse possível, no arquivo *“data_reader.py”*, foram feitas modificações para que, caso o parâmetro citado seja utilizado, o arquivo “portuguese_winogrande.txt” (no caminho “/data/portuguese”), que possui os esquemas do WINOGRANDE traduzidos para português, seja lido corretamente. Além disso, definiu-se como teste de avaliação para este treinamento o WSC, sendo esta a avaliação realizada a cada 200 iterações, como citado em 6.1.4.2.

Para execução do treinamento para ajuste fino, o comando de execução é um pouco diferente do apresentado na seção 6.1., como pode ser visto no comando de exemplo abaixo:

```
python main.py \
      --task_name portuguese-winogrande \
      --do_eval \
      --do_train \
      --eval_batch_size 10 \
      --data_dir "data/portuguese" \
      --bert_model bert-base-multilingual \
      --max_seq_length 128 \
      --train_batch_size 4 \
      --alpha_param 20 \
      --beta_param 0.2 \
      --learning_rate 5.0e-6 \
      --num_train_epochs 5 \
      --output_dir model_output/
```

O primeiro ponto de atenção neste comando é o parâmetro “--task _name”, que é utilizado como “portuguese-winogrande”. A partir de modificações no código, criou-se esse *task_name* de modo a possibilitar a leitura correta do *dataset* do WINOGRANDE traduzido usado no ajuste fino. Ambos os parâmetros “--do_eval” e “--do_train” são utilizados pois além de fazer o treinamento, ao final deste o programa realiza uma avaliação com o modelo com melhor acurácia (isso facilita para o usuário pois é possível visualizar a avaliação do melhor modelo ao fim da

execução). Os parâmetros “--eval_batch_size”, “--data_dir”, “--bert_model”, “--max_seq_length” e “--output_dir” são similares aos já apresentados em 6.1.4.1.

O parâmetro “--train_batch_size” representa o tamanho do lote (*batch*) utilizado no treinamento. Neste código, esse valor representa quantos esquemas serão lidos a cada iteração do modelo antes dele atualizar os seus parâmetros internos. Quanto menor este número, mais o modelo se adapta aos esquemas do *dataset* de treinamento. Porém, quanto menor o valor, mais passos de treinamento ocorrem por época e, por isso, mais tempo demora cada época de treinamento.

Os parâmetros “--alpha_param” e “--beta_param” são parâmetros comuns em treinamentos de *Machine Learning* e foram mantidos com valor padrão citado pelo autor (20 e 0.2, respectivamente). O “--learning_rate” é a taxa de aprendizado do modelo, crucial no cálculo do gradiente descendente citado em 6.1.4.2., definindo a velocidade de convergência do cálculo da perda. Este valor também foi mantido como o padrão fornecido pelo autor, de $5 \cdot 10^{-6}$ (5.0e-6).

O número de épocas de treinamento (*training epochs)*, denotado pelo parâmetro “--num_train_epochs” é o número de passagens completas pelo *dataset* de treinamento, sendo importante testar diversas possibilidades para esse valor para tentar encontrar o seu valor ótimo. Vale ressaltar que o tempo de execução de cada época é normalmente similar, portanto o tempo de execução total é, aproximadamente, o tempo de execução de uma época multiplicado pelo número de épocas de treinamento.

6.2.3. BERT Multilingual

O BERT Multilingual é uma versão do BERT, também disponibilizada gratuitamente pelo *Google*, em [7], que é compatível com 104 idiomas, incluindo o português, tendo sido pré treinado na Wikipédia desses idiomas na tarefa de *Masked Language Modeling* (MLM - Modelamento de Linguagem Mascarada). Este modelo está disponível apenas na arquitetura Base, tendo 12 camadas (blocos de *Transformers*), 12 *attention heads* e 110 milhões de parâmetros, assim como o BERT Base em inglês. Por ser multi-idiomas, espera-se que o desempenho em um dado idioma seja pior que o desempenho de um modelo pré treinado exclusivamente naquele idioma. Porém, como será realizado o ajuste fino no

modelo a partir do WINOGRANDE em português, o seu uso neste trabalho é uma opção bastante válida.

O BERT Multilingual já estava pré-configurado como um dos modelos possíveis no código utilizado, de [10], portanto não foram necessários ajustes para executar o modelo, exceto pelos ajustes de compatibilização dos dados do WINOGRANDE traduzido que serão usados para ajuste fino, como citado em 6.2.2.

6.2.3.1. Execução do BERT Multilingual sem ajuste fino

Antes de iniciar o processo de treinamento de ajuste fino, é importante executar o modelo sem passar por este processo, de modo a obter a sua acurácia sem ajuste fino, para que seja possível fazer a comparação dos desempenhos.

Executando o BERT Multilingual sem ajuste fino, atingiu-se uma acurácia de 58,6% no dataset do WSC273 em português. Apesar de ser um valor inferior ao obtido com o BERT Large sem ajuste fino na abordagem 1, o resultado obtido é bastante animador, pois supera os resultados alcançados por [2] e, além disso, espera-se que, com o ajuste fino a partir do WINOGRANDE traduzido, sejam alcançados desempenhos superiores ao alcançado sem ajuste fino.

6.2.3.2. Execução do BERT Multilingual com ajuste fino

Para o treinamento para ajuste fino do BERT Multilingual, utilizou-se a mesma estrutura de código utilizado por todo o trabalho, explicado em 6.1.4.2. Nesta seção, a etapa de treinamento citada será utilizada. As adaptações realizadas no código para esta etapa são citadas na seção 6.2.2.

Inicialmente, para teste e entendimento do processo de treinamento de ajuste fino, o modelo foi treinado em 800 esquemas (junção dos datasets *train_xs.jsonl* e *train_s.json* traduzidos), sendo posteriormente treinado no *dataset* completo do WINOGRANDE traduzido (obtido a partir do arquivo *train_xl.jsonl* traduzido). Uma boa parte do trabalho nesta etapa é a configuração dos hiperparâmetros de treinamento, principalmente o número de épocas de treinamento (quantas vezes se passa por completo pelo *dataset* de treinamento) e o *batch-size* (o número de esquemas que será lido para cada atualização dos parâmetros do modelo). Os outros parâmetros são mantidos com o valor padrão

fornecido pelo autor. É importante balancear esses parâmetros de modo que o tempo de execução não seja tão longo (se enquadrando no tempo de execução máximo do *Google Colab*, de 12 horas) e que a melhora na acurácia seja satisfatória.

Primeiramente, vamos comparar o desempenho após uma época de treinamento de ajuste fino, a partir do WINOGRANDE traduzido de inglês para português, para diferentes valores de *batch-size*. Vale ressaltar que o valor do *batch-size* está intimamente relacionado à memória disponível durante o processamento, pois todos os esquemas do lote são armazenados na memória RAM. Experimentalmente percebeu-se que o sistema de *runtime* do *Colab* suporta até um *batch-size* de 16. Neste primeiro comparativo iremos realizar o treinamento para *batch-sizes* 1, 4, 8 e 16, de forma a comparar o desempenho do modelo após o ajuste fino e o tempo de execução de cada treinamento. Os resultados obtidos são mostrados abaixo, na Tabela 6.6:

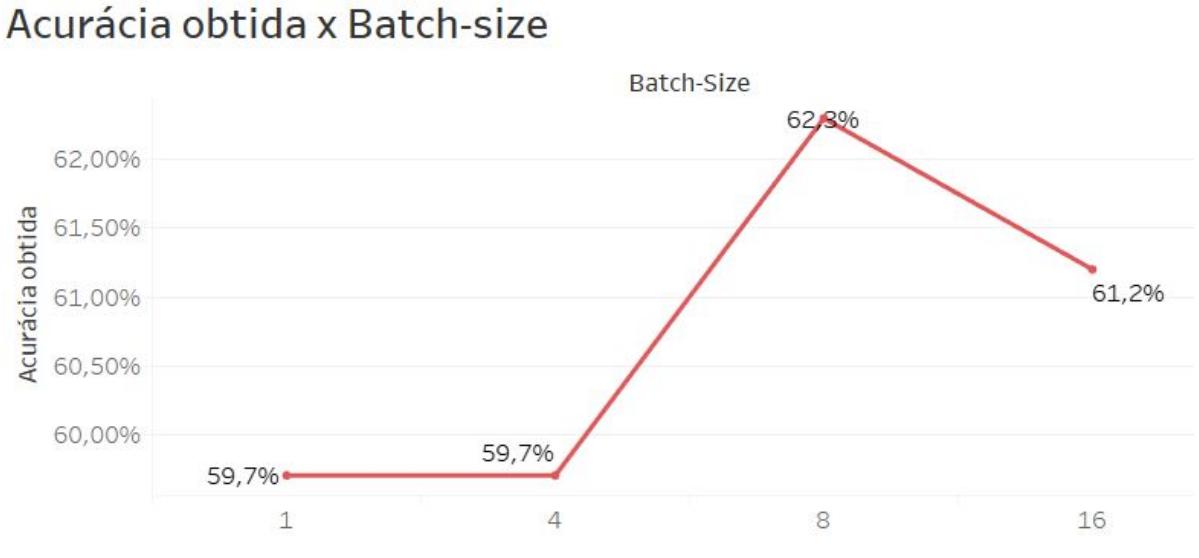
Tabela 6.6: Comparação entre as acurácias obtidas e os tempos de execução para o treinamento de ajuste fino por uma época, para diferentes tamanho de lote (*batch-size)*.

1 época de treinamento de ajuste fino		
Batch-Size	Acurácia obtida	Tempo de Execução (em horas)
1	59,7%	7,61
4	59,7%	2,75
8	62,3%	1,93
16	61,2%	1,57

Vale ressaltar que o *hardware* fornecido pelo *Google* para execução do *Google Colab* é variável, dependendo do momento do acesso. Assim, o tempo de execução pode variar dependendo do *hardware* alocado no momento do treinamento para ajuste fino.

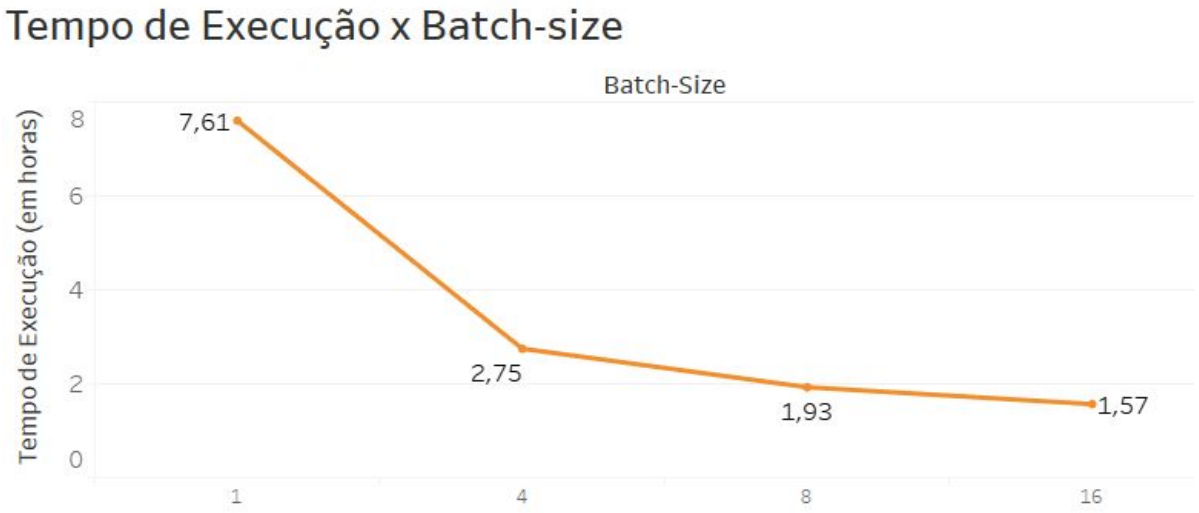
Podemos visualizar na figura 6.15 um gráfico que relaciona a acurácia obtida com o *batch-size* e, na figura 6.16 um gráfico que relaciona o tempo de execução de uma época de treinamento para cada *batch-size.*

Figura 6.15: Gráfico da acurácia obtida em função do *batch-size*, com uma época de treinamento de ajuste fino.



Fonte: Confecção própria.

Figura 6.16: Gráfico da acurácia obtida em função do *batch-size*, com uma época de treinamento de ajuste fino.



Fonte: Confecção própria.

Para um programador, a utilização de grandes *batch-sizes* é uma alternativa bastante atrativa pois, como visto, o tempo de execução de cada época é bastante

reduzido com o aumento deste valor. Porém, esta opção traz duas complicações: maiores *batch-sizes* requerem mais memória (como já dito, neste trabalho, devido às condições de execução no *Google Colab,* o *batch-size* fica limitado a, no máximo, 16) e para altos valores (64, 128 e superiores) o treinamento de ajuste fino acaba não conseguindo obter uma boa generalização do *dataset* de treinamento, pois ocorre o chamado sub-ajuste (*under-fitting*), no qual o modelo não teve iterações suficientes para conseguir obter uma boa generalização para ser aplicada nos dados de avaliação, o que implica em resultados não satisfatórios. Por outro lado, a utilização de *batch-sizes* muito pequenos (1 ou 4, como visto) faz com que o modelo atualize seus parâmetros a cada poucas leituras dos dados do *dataset* de treinamento. Isso também afeta a generalização do modelo pois, neste caso, o modelo adapta seus parâmetros aos dados de treinamento de forma excessiva, o que ocasiona o problema de sobreajuste (*over-fitting*), no qual o modelo passa a se sair muito bem nos dados de treinamento mas não nos dados de avaliação. Além disso, a redução do *batch-size* faz com que o tempo de execução de cada época seja aumentado.

Desta forma, é importante encontrar, experimentalmente, um valor de *batch-size* que seja intermediário entre os casos de *under-fitting* e *over-fitting*, obtendo um modelo que consiga generalizar os dados de treinamento de modo a obter uma boa acurácia no *dataset* de avaliação. Na análise experimental feita, pode-se perceber que o valor ótimo encontrado foi o treinamento para ajuste fino com *batch-size* 8, pois obtém a melhor acurácia em um tempo de execução aceitável. Podemos notar que o desempenho com *batch-sizes* menores (1 e 4) foi inferior, indicando o acontecimento de *over-fitting* e o desempenho com *batch-size* maior (16) também foi inferior, indicando o acontecimento de *under-fitting*. Caso fosse possível executar o código com *batch-sizes* ainda maiores (32, por exemplo), esse efeito seria ainda mais claro.

Tendo feita a análise da influência do *batch-size* na acurácia obtida e no tempo de execução, é importante fazer a mesma análise para o número de épocas de treinamento, mantendo um *batch-size* constante. Este processo é mais simples pois basta uma execução do treinamento, com um grande número de épocas. Porém, para a comparação, é necessário inserir um trecho de código que

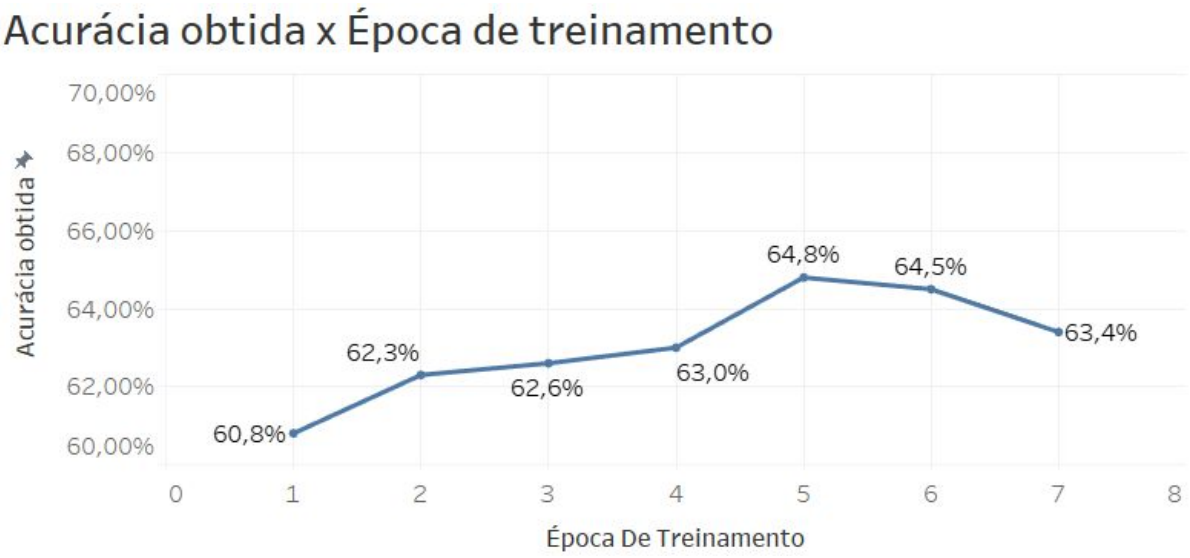
armazena a melhor acurácia obtida para cada uma das épocas de treinamento. Para possibilitar um número de épocas de treinamento grande (7 épocas), utilizou-se *batch-size* igual a 16 pois, como visto anteriormente, cada época é executada mais rapidamente (em torno de 1,5 hora por época) e as 7 épocas podem ser executados no tempo limite de execução do *Google Colab*, de 12 horas. Vale ressaltar que, neste caso, não faz sentido fazer a análise do tempo de execução, pois o tempo de execução de cada época é similar para todas as épocas. Os resultados podem ser visualizados na tabela 6.7:

Tabela 6.7: Comparação entre as acurácias obtidas para o treinamento de ajuste fino *batch-size* 16, para 7 épocas de treinamento.

Treinamento de ajuste fino com *batch-size* 16	
Época de treinamento	Acurácia obtida
1	60,8%
2	62,3%
3	62,6%
4	63,0%
5	64,8%
6	64,5%
7	63,4%

Na figura 6.17 é possível ver graficamente a evolução da acurácia obtida durante as épocas de treinamento.

Figura 6.17: Gráfico da acurácia obtida em função da época de treinamentos, com *batch-size 16.*



Fonte: Confecção própria.

Assim como visto em relação à influência do *batch-size* na acurácia obtida nos treinamentos de ajuste fino, o número de épocas de treinamento também pode gerar "*over-fitting*" e "*under-fitting*". Como visto na Figura 6.17, para poucas épocas de treinamento a acurácia é menor que para um número maior de épocas, pois nestes casos o modelo passou poucas vezes por completo sobre o *dataset* de treinamento e, por isso, ainda possui margem para melhorar a generalização, caracterizando o *under-fitting*. Percebe-se também, que a partir de um certo número de épocas a acurácia começa a cair, visto que ao passar muitas vezes pelo *dataset* de treinamento, o modelo passa a ficar muito adaptado àquele *dataset* específico, reduzindo a sua capacidade de generalização, caracterizando o "*over-fitting*". Devido às limitações do *Google Colab*, não foi possível realizar o treinamento com um número maior de épocas, o que ilustraria melhor este comportamento. Felizmente, na implementação utilizada não é necessário realizar diversas execuções buscando o número de épocas ótimo, pois o código faz avaliações periódicas da acurácia do modelo em todas as épocas, sempre armazenando o melhor modelo obtido e substituindo-o caso melhores precisões sejam alcançadas. Desta forma, independente da época em que a melhor precisão é obtida, o modelo com parâmetros que obtiveram esta precisão é armazenado e pode ser posteriormente utilizado.

Essas análises foram muito importantes para se chegar aos parâmetros para a execução que obteve os melhores resultados dentro das limitações do ambiente, já estabelecidas. Ainda assim, foi importante fazer uma análise experimental com algumas combinações de parâmetros até se obter esse resultado.

Os melhores resultados obtidos para o treinamento de ajuste fino do BERT Multilingual, a partir do *dataset* WINOGRANDE traduzido de inglês para português, foram alcançados executando o treinamento com *batch-size* 8, por 6 épocas de treinamento. Neste treinamento de ajuste fino, alcançou-se uma acurácia de 65,9%, após, aproximadamente, 9 horas de execução. Essa acurácia foi obtida na quinta época de treinamento. Este resultado foi bastante satisfatório, superando em mais de 7% (indo de 58,6% para 65,9%) a acurácia obtida sem o ajuste fino.

6.2.4. BERT Portuguese

O BERT possui uma versão, pré-treinada em um *corpus* em português (Wikipedia brasileira e o *corpus* brWaC, um grande *corpus* em português brasileiro obtido a partir de *crawling* da internet), disponibilizado gratuitamente pela companhia *NeuralMind*, em [8]. Foi necessário fazer o download do modelo e então incluí-lo no código. A inclusão foi bem simples, bastando apenas algumas adaptações no código, citadas anteriormente.

Duas versões do modelo são disponibilizadas. A versão Base, cujo nome é "*bert-base-portuguese-cased*" possui 12 camadas (blocos de Transformers), 12 attention heads e 110 milhões de parâmetros, como na versão Base original do BERT. A versão Large também é disponibilizada, com nome "*bert-large-portuguese-cased*" tendo, assim como no BERT Large original, 24 camadas (blocos de *Transformers*), 16 *attention heads* e 340 milhões de parâmetros. Vale ressaltar que além do modelo em si, também é disponibilizado o vocabulário que será usado no processo de *Tokenização*. O pré-treinamento desses modelos foi feito seguindo as recomendações do BERT original, de [7], nas duas tarefas citadas em 2.5.2.

6.2.4.1. Execução do BERT Portuguese sem ajuste fino

Antes de iniciar o processo de ajuste fino, vamos executar ambos os modelos no WSC273 em português para comparação. O modelo BERT Base Portuguese (*bert-base-portuguese-cased*) obteve uma acurácia de 54,9% no WSC273 em português. Já o modelo BERT Large Portuguese (*bert-large-portuguese-cased*) obteve uma ligeira melhora na acurácia, chegando a 55,3%.

Pode-se notar que os resultados obtidos por esses modelos sem ajuste fino são inferiores ao resultado obtido com o BERT Multilingual sem ajuste fino, mas ainda assim, são superiores aos resultados atuais do estado da arte em português [2]. Porém, apesar desta diferença entre os modelos Multilingual e Portuguese no teste sem ajuste fino, espera-se que os modelos em português obtenham resultado melhor após o ajuste fino, pois supõe-se que, dado que o pré-treinamento foi feito apenas em português, o modelo consegue desenvolver uma representação interna da língua portuguesa mais adequada do que o modelo Multilingual, que teve pré-treinamento em diversas línguas.

6.2.4.2. Execução do BERT Portuguese com ajuste fino

O próximo passo, como feito para o BERT Multilingual, para os modelos do BERT em português seria realizar o treinamento de ajuste fino dos modelos a partir do WINOGRANDE traduzido, de modo a buscar melhores acurácias no WSC273 em português.

Porém, após diversos testes de treinamento de ajuste fino com diferentes parâmetros, ambos os modelos apresentaram problemas no treinamento, passando pelo processo mas não atualizando os seus parâmetros internos. Com isso, a acurácia no WSC273 não é afetada pelo treinamento, mantendo os valores obtidos na execução sem ajuste fino. Este problema foi relatado por outras pessoas que estão trabalhando com esses modelos, mas uma solução ainda não foi disponibilizada. Supõe-se que esta seja uma limitação do modelo disponibilizado pela *Neuralmind*.

Deste modo, devido à limitação destes modelos, não é possível comparar o desempenho após ajuste fino com o modelo BERT Multilingual. Acredita-se que, se fosse possível realizar esta tarefa, os modelos com pré-treinamento exclusivo em português teriam melhor desempenho após ajuste fino, pois devido à exclusividade do pré-treinamento, a representação interna da língua no modelo seria mais adequada. Além disso, o modelo Large, com mais camadas em sua rede neural e mais parâmetros internos, possui melhor capacidade de adaptação aos dados de treinamento, devido à arquitetura citada. Assim, caso fosse possível realizar o ajuste fino, esse modelo provavelmente seria o de melhor desempenho.

7. Análise de Resultados

De forma a compilar os resultados obtidos apresentam-se nas tabelas 7.1 e 7.2 os resultados obtidos nas abordagens 1 e 2, respectivamente. Além disso, para tornar estes resultados mais acessíveis para entendimento e análise, criou-se um portal para a demonstração prática, detalhado no apêndice A.

Tabela 7.1: Resultados obtidos a partir da Abordagem 1.

Modelo	Dataset	Acurácia obtida
BERT-Large original	WSC273 traduzido	61,9%
BERT_WIKI_WSCR	WSC273 traduzido	72,9%

Tabela 7.2: Resultados obtidos a partir da Abordagem 2.

Modelo	Dataset	Acurácia obtida
BERT-Base Multilingual original	WSC273 português	58,6%
BERT-Base Portuguese original	WSC273 português	54,9%
BERT-Large Portuguese original	WSC273 português	55,3%
BERT-Base Multilingual WINOGRANDE traduzido	WSC273 português	65,9%

Uma conclusão bastante clara alcançada a partir dos dados apresentados é relativa à vantagem na utilização do BERT após treinamento de ajuste fino em um *dataset* de estrutura similar à estrutura do *dataset* de avaliação. Apesar do desempenho ser satisfatório sem ajuste fino, por conta de uma das tarefas de pré-treinamento do modelo ser similar à tarefa referente a este trabalho (*Masked-Language Modeling*), já obtendo resultados melhores que o estado da arte em ambas as abordagens, pode-se perceber que realizando o treinamento de ajuste fino obtêm-se melhorias na acurácia bastante significativas, da ordem de 10% quando comparados à execução dos modelos sem ajustes finos.

Na abordagem 1, notou-se que tanto na execução do BERT-Large sem ajuste fino, como na execução do BERT-Large com ajuste fino no WSCR, os resultados foram bastante similares no *dataset* original do WSC273 e no *dataset do* WSC273 traduzido de português para inglês. Mesmo levando em conta os erros na tradução e mascaramento, como apresentado na seção 6.1.5, o resultado com ajuste fino no *dataset* traduzido foi bastante semelhante ao resultado no *dataset* original. Esses resultados comprovam a hipótese de que é possível a transferência de conhecimento entre modelos de PLN em diferentes idiomas. Desta forma, pode-se concluir que é uma opção bastante plausível em desafios de PLN que possuem resultados avançados em um dado idioma (normalmente inglês) traduzir a sua formulação em outros idiomas para esse idioma com resultados mais avançados, executando os modelos nesse idioma. No caso deste trabalho, caso o modelo estado da arte em inglês, de [5], fosse disponibilizado em código aberto, com o ajuste fino já realizado, essa abordagem poderia ser repetida para esse modelo, provavelmente obtendo resultados ainda melhores, aproximando-se da acurácia do estado da arte em inglês. Vale ressaltar que uma opção seria tentar reconstruir este modelo de [5], realizando o ajuste fino, porém, como a proposta da abordagem 1 era justamente testar a possibilidade de transferência de conhecimento entre idiomas, optou-se por utilizar um modelo com ajuste fino já realizado e disponível para *download*. Além disso, dada as limitações do ambiente utilizado para a execução, provavelmente não seria possível obter resultados similares a [5].

Vale ressaltar que o desempenho nessa primeira abordagem está diretamente relacionado à qualidade do sistema de tradução implementado, sendo que quanto melhor o sistema de tradução, mais próximas as frases traduzidas são das frases originais, de modo que os resultados são bastante próximos em ambos os *datasets*. O sistema utilizado neste trabalho, baseado na API de tradução do *Google Cloud*, se mostrou bastante adequado. Nos esquemas em si, a tradução se mostrou muito próxima da versão original das frases, obtendo um bom desempenho na métrica BLEU, como visto na seção 6.1.1. Os principais problemas na tradução aconteceram nos campos *"pronoun*" e "*snippet*", que eram pequenos trechos do esquema, e, por não ter o contexto da frase, o sistema não se saiu tão

bem (para estes casos, porém, nem mesmo um falante de ambas as línguas conseguiria fazer uma tradução adequada sem o contexto das frases). Assim, pode-se concluir que o sucesso dessa abordagem está diretamente relacionado à escolha acertada da API de tradução.

Em relação à abordagem 2, pôde-se perceber inicialmente, na análise sem ajuste fino, que o BERT Multilingual se saiu melhor que as versões Base e Large do BERT Portuguese. Diversas razões podem explicar essa diferença: os modelos são pré-treinados em *corpus* diferentes, a versão Multilingual é pré-treinada em diversos idiomas, enquanto que a versão Portuguese é pré-treinada apenas em português, dentre outras. Ainda assim, tanto o BERT Multilingual como o BERT Portuguese sem ajuste fino tiveram desempenho superior ao estado da arte na solução dos desafios de Winograd em português, o que já é bastante satisfatório.

Com ajuste fino, dentro das limitações do ambiente de desenvolvimento, pôde-se perceber um bom avanço na acurácia do BERT Multilingual, sendo uma melhora de, aproximadamente, 7% (indo de 58,6% para 65,9%), comparável à melhora a partir do ajuste fino obtida na abordagem 1. Este resultado permite concluir que a abordagem escolhida, baseada na tradução do *dataset* WINOGRANDE de inglês para português, foi eficaz, melhorando significativamente o desempenho do modelo. Assim como na abordagem 1, a eficiência deste ajuste fino está diretamente relacionado à qualidade do sistema de tradução implementado. Como já citado, a API escolhida para tradução se mostrou bastante adequada, obtendo bons resultados tanto na métrica de tradução BLEU como na abordagem 1, pois os resultados obtidos foram bastante similares aos resultados obtidos a partir do *dataset* original em inglês. Ainda assim, caso a tradução do WINOGRANDE fosse feita ou revisada manualmente (o que fugiu do escopo deste trabalho devido à grande quantidade de esquemas) os resultados obtidos provavelmente seriam superiores.

Infelizmente, como não foi possível realizar o ajuste fino das versões do modelo BERT Portuguese, a comparação entre modelos com ajuste fino não pôde ser feita. Porém, caso fosse possível realizar este procedimento de ajuste fino, dado que o BERT Portuguese, em suas duas versões, teve pré-treinamento realizado apenas em português, espera-se que esses modelos tivessem

desempenho superior ao BERT Multilingual, visto que os seus parâmetros são todos baseados no *corpus* em português, sendo que a versão *Large* do modelo, com mais camadas e parâmetros, provavelmente teria o melhor desempenho, dada a capacidade desta arquitetura se adaptar melhor aos dados de treinamento.

Conclui-se portanto, a partir dos resultados apresentados, que este trabalho alcançou seu objetivo, avançando o estado da arte na solução dos esquemas de Winograd em português brasileiro nas duas abordagens estudadas, tanto usando modelos de linguagem em inglês, traduzindo o desafio para este idioma, bem como usando modelos de linguagem em português, a partir do ajuste fino em um *dataset* com estrutura similar ao desafio, traduzido de inglês para português.

8. Conclusão

O desenvolvimento do presente trabalho utilizou conhecimentos de inteligência artificial e programação, majoritariamente, para construir soluções para o Desafio de Esquemas de Winograd em português brasileiro. Foi possível, a partir dos modelos BERT utilizados, com e sem ajuste fino e com diversas variações, comparar os resultados entre as diferentes abordagens para, então, comprovar hipóteses e evoluir o estado da arte na língua.

Apesar de a grande área em que se encontra o problema específico do desafio, da inteligência artificial, estar em alta dentro e fora do meio acadêmico, a subárea de processamento de linguagem natural ainda não tem um avanço significativo em português brasileiro comparável ao avanço em outras línguas, como o inglês, por exemplo. Nesse contexto, ao utilizar a primeira abordagem de solução no projeto, baseada na construção de um sistema de tradução do português para o inglês, aplicando o *dataset* WSC273 traduzido de português para inglês em modelos pré-treinados e com ajuste fino em inglês, com isso chegando a até 72,9% de acurácia (superando em quase 30% o estado da arte em português brasileiro), confirma-se a possibilidade de transferência de conhecimento entre as línguas para resolução de desafios de PLN como o desafio de esquemas de Winograd.

Já ao utilizar a segunda abordagem, aplicando o *dataset* com esquemas em português brasileiro em modelos pré-treinados em português, conseguiu-se alcançar acurácia de 65,9% realizando ajuste fino a partir do *dataset* WINOGRANDE traduzido de inglês para português, um pouco abaixo do resultado obtido com a segunda abordagem, sendo que tal diferença era esperada dado o avanço da língua inglesa anteriormente mencionado, as limitações do ambiente para o ajuste fino do modelo e a diferentes *datasets* utilizados para ajuste fino do BERT. Entretanto, ao comparar-se este resultado com o baseline em português brasileiro, de 45,5%, conclui-se que nesta abordagem também foi possível avançar o estado da arte na língua, neste caso usando modelos propriamente em português brasileiro.

Desta forma, trabalhos futuros podem continuar atuando para melhorar ainda mais esses resultados. Uma das possibilidades é realizar ajuste fino em modelos especificamente pré-treinados em português brasileiro (e não em modelos multi-idiomas, como realizado neste trabalho), sendo plausível realizar um novo pré-treinamento do BERT em português brasileiro, dado que o modelo pré-treinado em português utilizado não foi compatível com a execução do ajuste fino. A tradução manual (ou correção manual da tradução automática) do *dataset* WINOGRANDE, usado para ajuste fino, também pode trazer melhorias para os resultados.

Além disso, caso deseje-se avançar os resultados na primeira abordagem, modelos com acurácia melhor em inglês poderiam ser utilizados, porém, provavelmente teriam de ser preparados e treinados pois os modelos em inglês com acurácias melhores do que o modelo utilizado não estão disponíveis em código aberto. Por fim, apesar de o BERT ter se mostrado a solução adequada à data de publicação deste trabalho, novos modelos e abordagens podem surgir com o avanço das tecnologias em processamento de linguagem natural em inglês e em português que, eventualmente, podem apresentar soluções mais eficazes para o desafio de esquemas de Winograd.

Referências

[1] LEVESQUE, H. J.; DAVIS, E.; Morgenstern, L. **The winograd schema challenge**. In Proceedings of the Thirteenth International Conference on Principles of Knowledge. Representation and Reasoning, 2011, pages 552–561. AAAI Press.

[2] MELO, G. S.; IMAIZUMI, V. A.; COZMAN, F. G. **Winograd Schemas in Portuguese**. São Paulo. In: ENCONTRO NACIONAL DE INTELIGÊNCIA ARTIFICIAL E COMPUTACIONAL (ENIAC), 16. , 2019, Salvador. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019 . p. 787-798.

[3] TURING, A. M. **Computing Machinery and Intelligence**. Mind, Volume LIX, Issue 236, October 1950, pp. 433-460

[4] WIKIPEDIA. **Comunicado sobre um milhão de artigos**. 2018. Disponível em <https://pt.wikipedia.org/wiki/Wikip%C3%A9dia:Comunicado_sobre_um_milh%C3%A3o_de_artigos>. Acesso em: 25 mai. 2020.

[5] SAKAGUCHI, K. et al. **WINOGRANDE: An Adversarial Winograd Schema Challenge at Scale**. CoRR, abs/1907.10641, 2019.

[6] PAPINENI, K. et al. **BLEU: a Method for Automatic Evaluation of Machine Translation**. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002, pp. 311-318.

[7] DEVLIN, J. et al. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. CoRR, abs/1810.04805, 2018.

[8] SOUZA, F.; NOGUEIRA. R. F.; LOTUFO, R. A. **Portuguese Named Entity Recognition using BERT-CRF**. CoRR, abs/1909.10649, 2019.

[9] VASWANI, A. et al. **Attention is All You Need**. CoRR, abs/1706.03762, 2017.

[10] KOCIJAN V. et al. **A Surprisingly Robust Trick for the Winograd Schema Challenge**. CoRR, abs/1905.06290, 2019.

[11] MACEDO, Rodrigo. **Introdução a Processamento de Linguagem Natural**. Ensina.AI, 2018. Disponível em: <https://medium.com/ensina-ai/introdu%C3%A7%C3%A3o-a-processamento-de-linguagem-natural-174936c096b>. Acesso em: 12 abr. 2020.

[12] COELHO, Renata. **Natural Language Processing. Um pouco de história.** IBM, 2019. Disponível em: <https://www.ibm.com/developerworks/br/library/natural-language-processing-um-pouco-de-historia/index.html>. Acesso em: 12 abr. 2020.

APÊNDICE A - Documentação do portal para demonstração prática

A fim de obter meios de comparar, interativamente, o resultado dos modelos em uma demonstração prática, criou-se um *website*[3] com funcionalidades simples, cujo objetivo é o de permitir que qualquer pessoa possa acessar e realizar testes com esquemas e seus respectivos resultados para cada modelo.

Para tanto, utilizou-se o *GitHub Pages* por possuir hospedagem gratuita do *website* diretamente do repositório, sem a necessidade de realizar um *deploy*, simplificando o processo de construção do mesmo. A página principal, “Projeto”, possui breve introdução sobre o tema, alguns exemplos de esquemas ambíguos de Winograd e apresentação dos autores.

Sabe-se que demonstrar o treinamento de um modelo seria inviável dado o tempo de processamento, que pode chegar em até 12 horas no caso do ajuste fino e, mesmo no caso de demonstração apenas da avaliação do modelo, apesar de ter um tempo de processamento muito menor, o fato de serem muitos modelos e testes torna essa demonstração exaustiva e longa.

Tendo isso em vista, optou-se por criar um *dataset* contendo informações sobre os esquemas como:

- Esquemas em português contendo sentença, candidatos e candidato certo
- Esquemas traduzidos para o inglês contendo sentença, candidatos e candidato certo
- Esquemas originais contendo sentença, candidatos e candidato certo

Além disso, o *dataset* contém os resultados que o modelo chegou para cada sentença, ou seja, existe uma coluna para cada modelo com seus respectivos resultados (acerto/erro), em linha com as sentenças. De posse dos dados, encontrou-se uma forma de criar um *dashboard* interativo que permite a filtragem de esquemas e escolhas de diferentes abordagens de solução, apresentando os resultados.

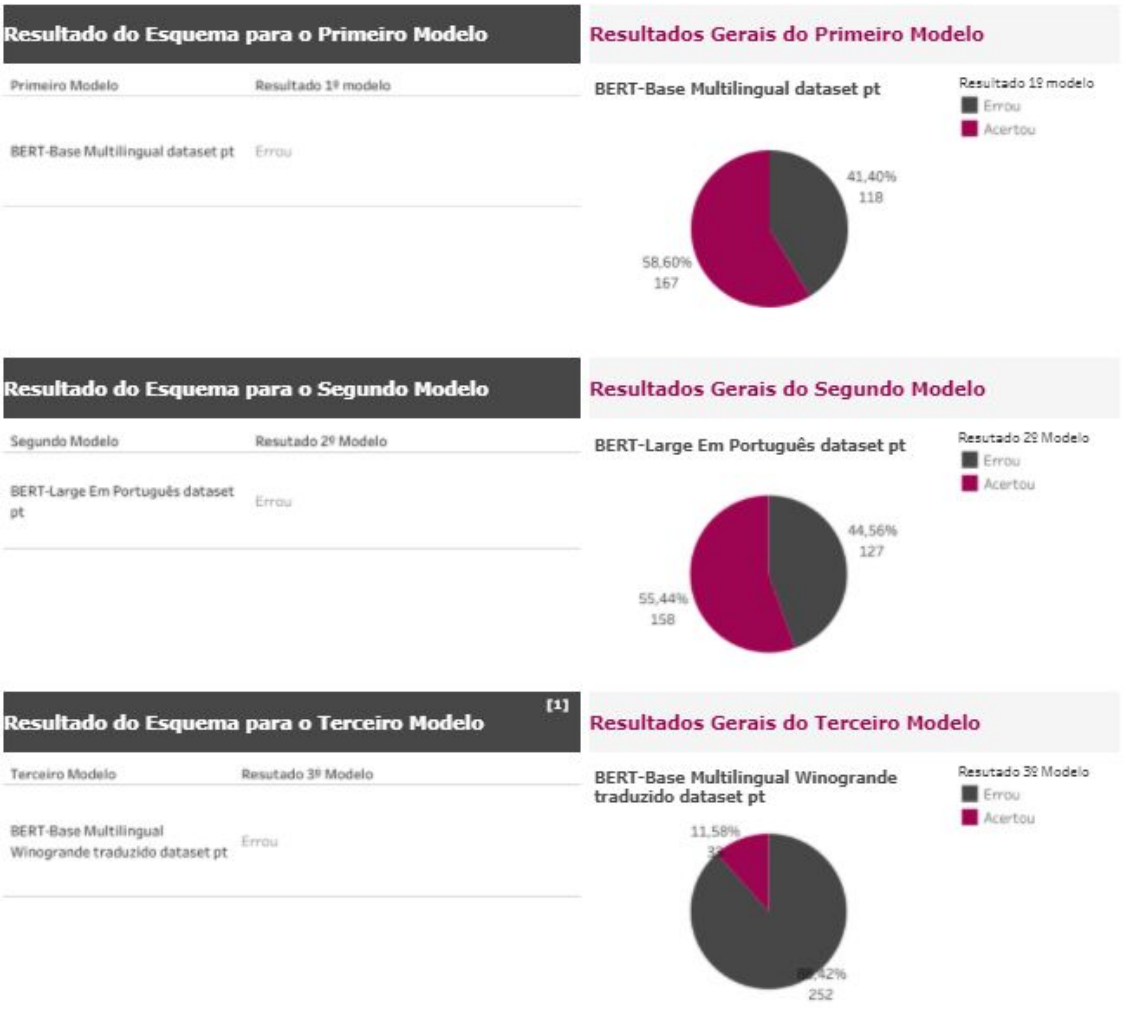
3. Desafio de Esquemas de Winograd. Disponível em <https://thaapontes.github.io/tcc/> Acesso em: 16 nov. 2020.

Essa forma de apresentação dos dados baseou-se no *Tableau Public*, ferramenta que facilita a publicação de visualizações em modo público em *websites*. Tal ferramenta possibilitou que tal *dashboard* fosse construído e compartilhado publicamente no *website* descrito anteriormente. Na figura A.1 abaixo, pode-se observar a possibilidade de selecionar um esquema em português, cuja funcionalidade é filtrar todas as visualizações abaixo de acordo com o esquema escolhido, bem como a abordagem, que é filtrada, por outro lado, nas visualizações da figura A.2:

Figura A.1: Esquemas em português, traduzidos, originais e seus respectivos candidatos e candidatos certos.

Fonte: Captura de tela da Demonstração Prática[3]

Figura A.2: Resultado dos modelos, a depender da abordagem, para o esquema escolhido.



Fonte: Captura de tela da Demonstração Prática[3]