unesp
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

Matheus Santi dos Santos

Geração de Questões Utilizando *Transformers* para Sistemas Tutores Inteligentes

São José do Rio Preto
2023

Matheus Santi dos Santos

Geração de Questões Utilizando *Transformers* para Sistemas Tutores Inteligentes

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto

Orientador:

Prof. Dr. Aleardo Manacero Jr.

São José do Rio Preto
2023

S237g

Santos, Matheus Santi Dos

Geração de Questões Utilizando Transformers para Sistemas Tutores Inteligentes / Matheus Santi Dos Santos. -- São José do Rio Preto, 2023

95 f.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto

Orientador: Aleardo Manacero Jr.

1. Ciência da computação. 2. Sistemas de computação. 3. Redes neurais (Computação). 4. Inteligência artificial Aplicações educacionais. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Matheus Santi dos Santos

Geração de Questões Utilizando *Transformers* para Sistemas Tutores Inteligentes

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto

Comissão Examinadora:

Prof. Dr. Aleardo Manacero Jr.
UNESP - Câmpus de São José do Rio Preto
Orientador

Prof. Dr. Caio Saraiva Coneglian
Unimar - Universidade de Marília

Prof. Dr. Arnaldo Candido Junior
UNESP - Câmpus de São José do Rio Preto

São José do Rio Preto
5 de setembro 2023

Aos meus pais Débora e Joaquim.

Aos meus avós Lomar e Antônio.

Agradecimentos

Agradeço a Deus por ter me dado a vida, a saúde e a sabedoria necessárias para realizar este trabalho, me guiando durante o percurso até aqui e me permitindo conquistar mais essa realização em minha vida.

Agradeço aos meus familiares, em especial aos pais, Débora e Joaquim, por terem sido minha base e por todo o suporte que me proporcionaram durante meus anos de estudo. Sem o amor, o carinho e a dedicação deles, eu não estaria aqui hoje.

Quero também agradecer a minha namorada Jéssica Vicentini por ter me apoiado, me compreendendo e me encorajando para superar os desafios deste trabalho e manter a motivação para alcançar meus objetivos.

Ao meu orientador, Prof. Dr. Aleardo Manacero Jr., pelos ensinamentos e direcionamentos que tornaram possível a concretização deste projeto e incrementaram meu crescimento pessoal e profissional.

Agradeço também aos companheiros do PPGCC, em especial a Fernanda Peronaglio, que me apoiaram e compartilharam seus conhecimentos e experiências.

Por fim, agradeço aos professores do programa e aos funcionários da instituição por toda ajuda e suporte.

Resumo

O modelo tradicional de ensino possibilitou a formação de milhões de pessoas ao longo dos anos, mesmo com diversos pontos negativos em seu funcionamento. Buscando solucionar esses pontos, novas metodologias de ensino surgiram, focando na individualização do ensino, em que este se adaptaria às formas que o aluno aprende de maneira mais eficaz. Esta individualização, no entanto, traz problemas num ambiente tradicional, principalmente ligados à necessidade de acompanhamento individual do aluno. Para solucionar este problema, Sistemas Tutores Inteligentes podem ser uma solução, utilizando diversas técnicas que permitam compreender as necessidades do aluno e auxiliá-lo com uma menor supervisão dos professores. No entanto, de acordo com a abordagem utilizada, ainda existe a necessidade de desenvolver os conteúdos que devem ser apresentados aos alunos. Para tornar este processo menos trabalhoso, formas automatizadas de criação de conteúdo podem ser exploradas, entre as quais a utilização de técnicas de geração de questões para diminuir o trabalho presente na adaptação de conteúdo textual para perfis de aprendizado que possam se beneficiar deste tipo de conteúdo. Dentre estas técnicas, a Geração de Questões pode ser aplicada para fornecer atividades que possibilitam explorar e validar o nível de conhecimento do aluno. A forma como a Geração de Questões ocorre é muito variada, mas, analisando o cenário atual, a utilização de redes neurais, principalmente as do tipo *transformers*, se mostraram a melhor abordagem. Estas redes possuem diversos mecanismos internos, destacando-se a utilização de cálculos de atenção para realizar a tarefa proposta, e são conhecidas pelo seu grande potencial na realização de tarefas que envolvam o processamento de linguagem natural e têm recebido grande atenção desde sua publicação, tendo diversos avanços em relação a sua arquitetura e formas de se realizar o treinamento destas redes. Para o cenário de sistemas tutores inteligentes foram propostas quatro arquiteturas diferentes, utilizando redes *transformers* em sua composição, permitindo avaliar a que melhor se encaixaria a este ambiente. A validação destas arquiteturas é realizada através da avaliação de questões e respostas geradas pelas mesmas, utilizando textos de assuntos diferentes para simular a aplicação de diversas áreas de aprendizado. Formas automatizadas de avaliação de questões geradas são um desafio presente na literatura e, portanto, não é possível realizá-las, mas, com a definição de uma abordagem qualitativa, é possível validar a qualidade das 69 questões geradas e definir a arquitetura Multi-Modelo com Gerador de Resposta como a que melhor se encaixa ao cenário proposto, por obter a melhor concentração de resultados, com 18 questões ótimas, 27 boas, 13 regulares e apenas 11 ruins. Com isto conclui-se que a utilização desta arquitetura ligada a um Sistema Tutor Inteligente é possível, apesar de não eliminar totalmente a necessidade de interação do professor, mas permitindo uma grande facilitação no momento de desenvolvimento de atividades.

Palavras-chaves: Geração de Questões. Sistemas Tutores Inteligentes. *Transformers*.

Abstract

The traditional education model has enabled the education of millions of people over the years, even with several negative aspects in its operation. In an effort to address these issues, new teaching methodologies have emerged, focusing on the individualization of education, where it would adapt to the ways in which the student learns more effectively. However, this individualization brings problems in a traditional environment, mainly related to the need for individual student supervision. To address this issue, Intelligent Tutoring Systems can be a solution, using various techniques that allow understanding the student's needs and assisting with less teacher supervision. However, depending on the approach used, there is still a need to develop the content that should be presented to the students. To make this process less labor-intensive, automated content creation methods can be explored, including the use of question generation techniques to reduce the work involved in adapting textual content to learning profiles that can benefit from this type of content. Among these techniques, Question Generation can be applied to provide activities that enable the exploration and validation of the student's level of knowledge. The way Question Generation occurs is quite varied, but, considering the current scenario, the use of neural networks, especially those of the "transformer" type, has proven to be the best approach. These networks have various internal mechanisms, with a particular emphasis on the use of attention calculations to perform the proposed task. They are known for their great potential in tasks involving natural language processing and have received significant attention since their publication, with several advancements in their architecture and training methods. In the context of intelligent tutoring systems, four different architectures using "transformer" networks have been proposed, allowing for the evaluation of which one would best fit this environment. The validation of these architectures is carried out through the assessment of questions and answers generated by them, using texts on different subjects to simulate the application of various learning areas. Automated assessment of generated questions is a challenge in the literature and, therefore, cannot be performed. However, with the definition of a qualitative approach, it is possible to validate the quality of the 69 questions generated and determine the Multi-Model with Response Generator architecture as the one that best fits the proposed scenario, as it achieves the best concentration of results, with 18 excellent questions, 27 good ones, 13 regular ones, and only 11 poor ones. With this, it is concluded that the use of this architecture in conjunction with an Intelligent Tutoring System is possible, although it does not completely eliminate the need for teacher interaction, but it greatly facilitates the development of activities.

Keywords: Question Generation. Intelligent Tutoring Systems. Transformers.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

STI	Sistema Tutor Inteligente
POS	Part Of Speech
NER	Named-Entity Recognition
IA	Inteligência Artificial
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Units
SQuAD	Stanford Question Answering Dataset
BLEU	Bilingual EvaLuation Understudy
GPT	Generative Pre-trained Transformer
BERT	Bidirectional Encoder Representations from transformers
GLUE	General Language Understanding Evaluation
T5	Text-to-Text Transfer Transformer
METEOR	Metric for Evaluation of Translation with Explicit ORdering
ROUGE	Recall-Oriented Understudy for Gisting Evaluation

Sumário

1 Introdução

O sistema de ensino presente na maioria das escolas e universidades seguem preceitos criados a séculos, muitas vezes denominados de modelo tradicional de ensino. Ao longo dos anos diversos pesquisadores vêm apresentando alternativas para este modelo, embora estes se mostrem muitas vezes pouco atrativos e de difícil aplicação.

Este cenário, no entanto, vem se modificando ao longo do tempo, principalmente com o surgimento de novas tecnologias, como é o caso dos Sistemas Tutores Inteligentes (STI). Esses sistemas podem ter, entre outras funcionalidades, a aplicação de um ensino adaptado para cada aluno, permitindo a atualização do modelo de ensino para um que faça um tratamento mais exclusivo de cada aluno.

1.1 Modelo Tradicional de Ensino

O Modelo Tradicional de Ensino é também conhecido como expositivo, nome que deriva de sua principal característica: o conhecimento é concentrado na figura do professor, que o expõe aos aprendizes. A função do professor é ser, então, o principal ator na disseminação do conhecimento para a turma, um grupo de alunos que pode variar de poucos alunos, especialmente para alunos de idade mais baixa, a poucas centenas no caso de universidades.

A principal crítica a este modelo está relacionado à teoria de construção de conhecimento proposta por Piaget (1), que argumenta que o conhecimento é construído por meio da experiência e do processo de reflexão. No entanto, o Modelo Tradicional raramente incentiva a reflexão crítica e a participação dos alunos (2).

Além disso, o modelo tradicional ignora as percepções e preferências dos alunos pois, durante a aula, o professor discorre sobre um conteúdo determinado com poucas ou nenhuma interação dos alunos. Além disso, o conteúdo é apresentado de uma única forma, sendo esta a preferida pelo professor e tem pouca ou nenhuma flexibilidade em sua apresentação. Muitos educadores argumentam que essa rigidez é menos eficaz, uma vez que os alunos possuem diferentes características na forma de aprender, ou seja, possuem estilos de aprendizagem diferentes.

Em volta deste modelo reside outro problema: como avaliar os alunos. Avaliação é uma parte fundamental do processo, visto que, se busca classificar os alunos de acordo com sua capacidade de absorver o conteúdo apresentado pelo professor para identificar se o ensino tem sido efetivo. Geralmente as avaliações ocorrem em forma de provas ou atividades executadas pelos alunos. Os alunos que ficam abaixo de uma nota determinada são considerados reprovados e, dependendo da política da instituição de ensino, podem

realizar uma avaliação adicional, como um exame de recuperação, ou serem obrigados a frequentar esta aula novamente.

A avaliação nesta forma também pode não ser a mais eficaz, visto que promove entre os alunos a necessidade de memorizar e reproduzir os conteúdos (3), e não a de compreender e aplicar o conhecimento transmitido. Isso ocorre, por exemplo, com disciplinas que envolvam fatos históricos nas quais, de forma automatizada, os alunos acabam decorando datas importantes, muitas vezes sem compreender o real motivador por trás daqueles acontecimentos.

Por fim, a utilização deste modelo não leva em consideração algumas características essenciais na aprendizagem, como o conhecimento prévio do aluno ou a necessidade de uma abordagem diferenciada e individualizada para que o aluno absorva de forma mais eficiente o conhecimento.

De forma realista, tratar cada aluno individualmente é muito difícil no cenário do ensino atual, visto que é complicado para um professor apresentar conteúdos usando várias abordagens. A ideia de abordagens distintas envolve a classificação destes alunos em grupos que remetam ao seu estilo de aprendizagem, como é o caso do modelo de Kolb (4) em que os alunos são agrupados com base na forma como melhor absorvem o conhecimento.

Um possível auxílio na apresentação de conteúdos baseados em estilos de aprendizagem é a utilização de ferramentas digitais para o ensino, como é o caso dos Sistemas Tutores Inteligentes (STI), que permitem a aplicação destes modelos de aprendizagem de uma maneira mais eficaz e menos trabalhosa.

1.2 Sistemas Tutores Inteligentes

Com a evolução da tecnologia, os sistemas digitais tornaram-se cada vez mais presentes no dia a dia das pessoas, tanto facilitando o acesso à informação quanto automatizando ou agilizando processos como, por exemplo, acessar um conteúdo educacional de forma remota, sem precisar se deslocar até uma instituição de ensino. Na área de ensino foram criados diversos tipos de sistemas, que vão desde sistemas consultivos para os alunos verificarem suas notas, até sistemas completos com atividades disponíveis e correção automática das mesmas.

Dentre essas plataformas temos os Sistemas Tutores Inteligentes, que são sistemas automatizados capazes de proporcionar ao aluno uma experiência individualizada, sem a necessidade de interação constante do professor. Esses sistemas foram propostos por Carbonell em 1970 (5), sendo cada vez mais adotados devido à evolução de algoritmos inteligentes que possibilitam a automação de processos contidos na educação. Tais sistemas possuem uma arquitetura consolidada, composta por quatro módulos independentes:

- Módulo de Conhecimento de Domínio ou Especialista é o módulo que concentra os conhecimentos que serão apresentados, possivelmente em formatos distintos, ao aluno nas diversas disciplinas oferecidas pelo sistema.
- O Módulo Diagnóstico do Aluno é responsável por avaliar extensivamente o aluno, englobando não só o desempenho nas atividades propostas, mas também as características do aluno, seus conhecimentos prévios sobre o assunto e até mesmo seu estado mental.
- O Módulo de Interface do Usuário é o módulo que intermedeia a interação entre o aluno e o sistema, compondo as diferentes formas que essa interação pode ser realizada, como sistemas web, aplicativos móveis ou notificações por e-mail ou sms.
- Por fim, o Módulo Tutor é responsável por realizar as ações que um professor realizaria em sala de aula, mas individualmente, propondo atividades e sugestões para o aluno, além de utilizar as informações contidas nos demais módulos, como gerar atividades de acordo com o perfil do aluno estipulado pelo módulo de diagnóstico.

A utilização destes sistemas pode contribuir para a formação de alunos de todas as idades, permitindo uma forma de ensino mais individualizada e transformando gradualmente nossa sociedade e sociedades ao redor do mundo em uma sociedade mais educada.

Para que esse sistema funcione, quando aplicado em uma instituição de ensino já existente, há a necessidade de adequar o conteúdo ao sistema, tornando seu desenvolvimento e implementação mais onerosos e demorados. Além disso, como o sistema pode tratar cada aluno de forma diferenciada, há também a necessidade de construir novas atividades para ocasiões atípicas, tornando o processo interminável e transformando o professor, detentor do conhecimento, em um gerador de atividades.

Para lidar com esses problemas formas automatizadas de geração de conteúdo podem ser exploradas. Nelas seria necessário incluir apenas a origem da informação, aplicando técnicas como sumarização de textos ou buscas de materiais relacionados. Além disso, podem ser aplicadas técnicas capazes de gerar a própria atividade relacionada a esta informação.

1.3 Objetivo

Como evidenciado, Sistemas Tutores Inteligentes podem auxiliar educadores na utilização de abordagens que individualizam o aluno. Isso permite uma melhor adequação do conteúdo ao aluno, fazendo com que este seja apresentado da forma que o aluno tenha mais facilidade em aprender.

STIs são sistemas complexos e por isso podem ser subdivididos em diversas camadas diferentes, sendo assim, possível desenvolver cada uma dessas partes separadamente. Vi-

sando isto, este trabalho tem relação com demais trabalhos correlatos (6) que possuem como objetivo compor um Sistema Tutor Inteligente completo.

Neste cenário este trabalho teve como objetivo suprir a necessidade da formulação de atividades que STIs possuem, especificamente no que diz respeito a questões relacionadas ao conteúdo textual apresentado. Mais precisamente, o objetivo foi permitir a geração de questões que serão utilizadas nos estilos de aprendizados que possam se beneficiar deste tipo de atividade.

Com este objetivo em mente, os seguintes passos foram executados para que se chegasse ao resultado apresentado:

1. Explorar a literatura em busca de técnicas, como os Transformers (7), que possam ser utilizadas para geração de questões a partir de textos.

2. Definir entre essas técnicas, as que apresentam melhor aplicabilidade no cenário de Sistemas Tutores Inteligentes.

3. Propor arquiteturas que utilizam tais técnicas para a tarefa de Geração de Questões.

4. Validar estas arquiteturas, avaliando seus resultados e definir a que será utilizada no Sistema Tutor Inteligente em desenvolvimento.

1.4 Organização do texto

Este texto está organizado em seis capítulos, sendo que neste primeiro se fez a introdução das características e objetivos do projeto. No segundo capítulo é abordada a definição da tarefa de Geração de Questões e quais técnicas são regularmente encontradas na literatura. Já no terceiro capítulo se apresenta a arquitetura de redes neurais do tipo Transformer e suas variações. No quarto capítulo apresenta-se a proposta de solução e as arquiteturas desenvolvidas para se obter os resultados esperados. No quinto capítulo são apresentados os resultados obtidos por estas arquiteturas juntamente com uma interpretação destes. Por fim, no sexto capítulo são abordadas as conclusões e trabalhos futuros.

2 Geração de Questões

Como apresentado no capítulo 1, o objetivo deste trabalho esta ligado a abordagens que possam ser utilizadas na geração de questões, com o intuito de utilizá-las na composição de um sistema tutor inteligente que será capaz de realizar diversas etapas envolvidas no aprendizado, sendo destacada entre elas a formulação de atividades aos alunos, especificamente na geração de perguntas a serem respondidas com base no conhecimento apresentado. Portanto, é necessário que primeiramente se compreenda as características destas tarefas e como funcionam as suas relações.

2.1 Conceituação

O processamento de linguagem natural tem se tornado parte rotineira de sistemas computacionais, permitindo que a interface entre computador e seres humanos fique mais próxima da prática humana, como ocorre hoje em sistemas como o chatGPT (8) ou o Bard (9). Uma parte essencial no processamento de linguagem natural está na capacidade de um sistema realizar perguntas coerentes ao humano. Esta capacidade é posta a prova na utilização dos contextos informacionais para a geração de questionamentos que façam correlação ao mesmo. Um destes contextos informacionais está voltado para a interpretação de textos, em que a informação está inserida utilizando a linguagem natural e deve ser analisada por este sistema.

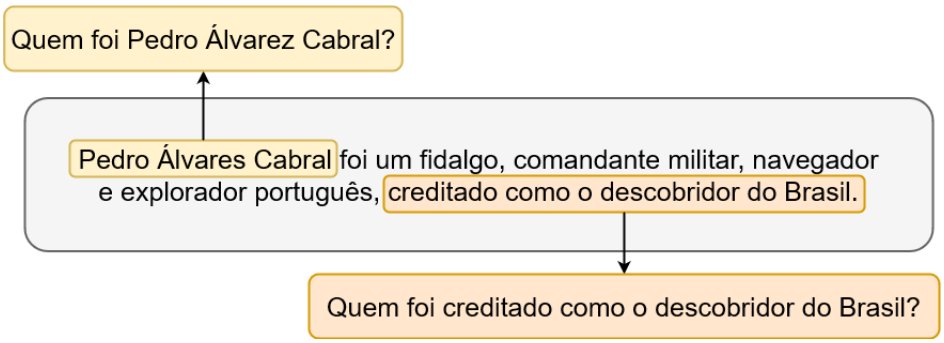
A Geração de Questões, do inglês *Question Generation* (QG) ou ainda *Natural Question Generation* (NQG), é a tarefa de, dada uma entrada informacional, gerar uma ou mais questões relacionadas à mesma (10). Diferentemente dos questionamentos realizados por interfaces para que ocorra confirmação de uma ação ou disponibilização de uma informação por parte do usuário, os sistemas de geração de questões têm sua utilidade voltada para a formulação da questão em si, buscando realizar tal processo sem a necessidade de interação externa.

Como dito anteriormente, tais questões podem ser geradas a partir de diversas fontes informacionais, variando de textos, foco deste trabalho, chegando até mesmo a utilização de vídeos e imagens (11).

O procedimento de geração de questões normalmente se dá através da identificação das informações contidas nesta fonte, realizando uma conversão desta fonte em questionamentos. Por exemplo, no texto mostrado na Figura 1, é possível extrair questões como “Quem foi Pedro Álvarez Cabral?” ou “Quem foi creditado como o descobridor do Brasil?”.

O processo de geração de questões ganhou força principalmente com o avanço das

Figura 1 – Exemplo de Geração de Questões



(Gerada pelo autor)

técnicas de inteligência artificial e do desempenho dos sistemas de hardware, que permitiram aos computadores compreenderem cada vez melhor a forma como os seres humanos se comunicam. Esta área de estudo é conhecida como Processamento de Linguagem Natural, sendo fortemente direcionada para possibilitar a comunicação entre máquina e humano na forma como este se comunica com os demais seres humanos.

Apesar de não possuir tanta visibilidade quanto técnicas que envolvem o processo contrário, como *Question Answering*, em que o intuito é a máquina fornecer informações ao homem, o paradigma de Geração de Questões se mostrou muito útil ao se considerar a necessidade de gerar bases de dados para o treinamento de algoritmos que busquem realizar a geração automática de respostas.

Outra aplicação explorada é a de auxiliar ou eliminar o trabalho presente nas escolas e universidades, em que o professor, como detentor do conhecimento, deve propor diversas questões para testar o conhecimento de seus alunos, o que é uma tarefa exaustiva e muitas vezes repetitiva. Atualmente ainda não se estabeleceu uma maneira eficiente de se realizar esta tarefa, sendo possível a utilização de abordagens totalmente diferentes umas das outras. Partindo deste pressuposto, por meio de uma revisão bibliográfica da literatura, foi realizado o levantamento das técnicas utilizadas neste paradigma, identificando três metodologias principais, que foram denominadas de Técnicas Semânticas e Sintáticas, Técnicas Semânticas e Sintáticas Apoiadas por I.A. e Técnicas de *Deep Learning*.

2.2 Técnicas Semânticas e Sintáticas

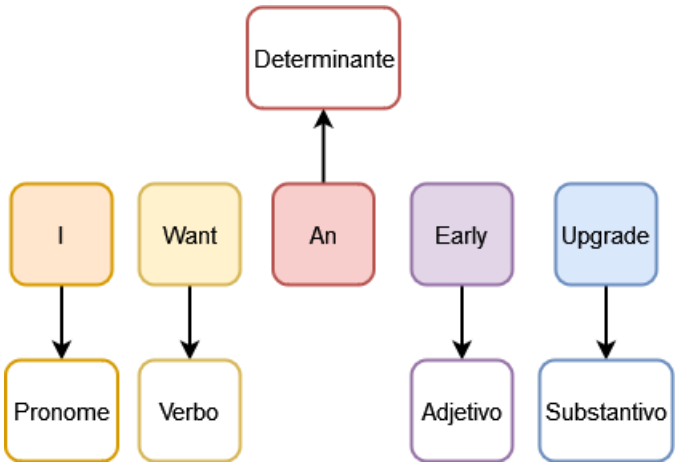
Nesta metodologia a classificação semântica e sintática das informações é a principal tarefa, utilizando regras de acordo com essas classificações para, de forma determinística, realizar a geração da questão.

Dentre as classificações observadas, três formas se destacam, sendo estas classes gramaticais (*Part of Speech*), reconhecimento de entidades nomeadas (*named-entity recognition*)

e análise de dependências (*Dependency Parsing*).

Part of Speech, ou POS, utiliza de algoritmos baseados em regras para demarcar na frase a classe gramatical de cada palavra, como demonstrada na Figura 2, permitindo a criação de regras específicas para cada tipo de palavra ou um conjunto das mesmas.

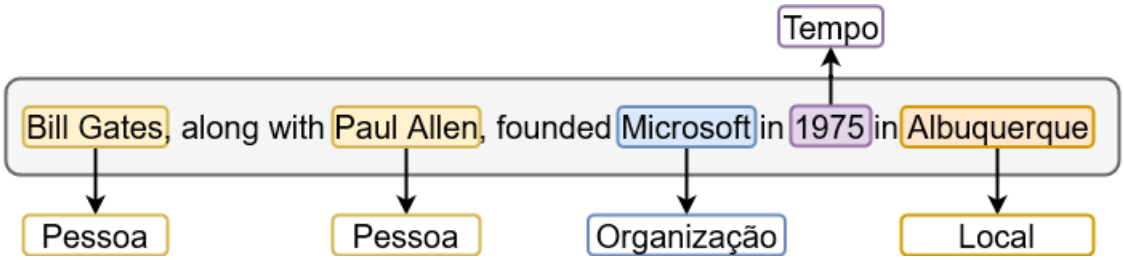
Figura 2 – Demarcação POS



(Gerado pelo autor)

De forma semelhante a POS, as demarcações de *named-entity recognition*, exemplificada na Figura 3 auxiliam no processamento do texto destacando na frase as entidades identificáveis, como locais, pessoas e organizações.

Figura 3 – Demarcação NER

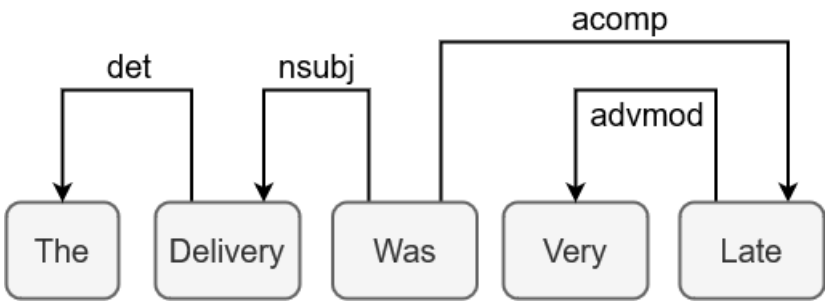


(Gerada pelo autor)

Por fim, as demarcações realizadas por algoritmos de *Dependency Parsing* contribuem no processamento gerando árvores de dependência, atribuindo para cada tipo de relação uma sigla específica, como na Figura 4, que podem ser interpretadas pelos algoritmos auxiliando na compreensão da frase.

Após tais etapas, as frases demarcadas passam para um algoritmo que as utiliza para gerar a questão, normalmente sendo implementados separadamente para cada tipo de demarcação, o que torna este um elemento importante em relação à qualidade do conteúdo gerado. Tais algoritmos normalmente fazem a utilização de modelos de questões para que,

Figura 4 – Demarcação Dependency Parsing



(Gerada pelo autor)

de acordo com regras pré-definidas, utilizem as demarcações para preencher as lacunas nos modelos, o que torna as questões repetitivas. A principal vantagem destes tipos de modelo está relacionada a previsibilidade dos resultados, por se tratarem de algoritmos sequenciais, diferente das outras abordagens.

Uma tendência observada nos trabalhos que aplicaram tais técnicas foi a tentativa de utilização de algoritmos avaliadores das questões geradas, a fim de que, após o processamento dos algoritmos, gerando diversas questões diferentes, as questões que não tivessem sentido semântico ou sintático fossem removidas. Baseado nos trabalhos analisados, uma arquitetura deste tipo de técnica seguiria os aspectos contidos na Figura 5.

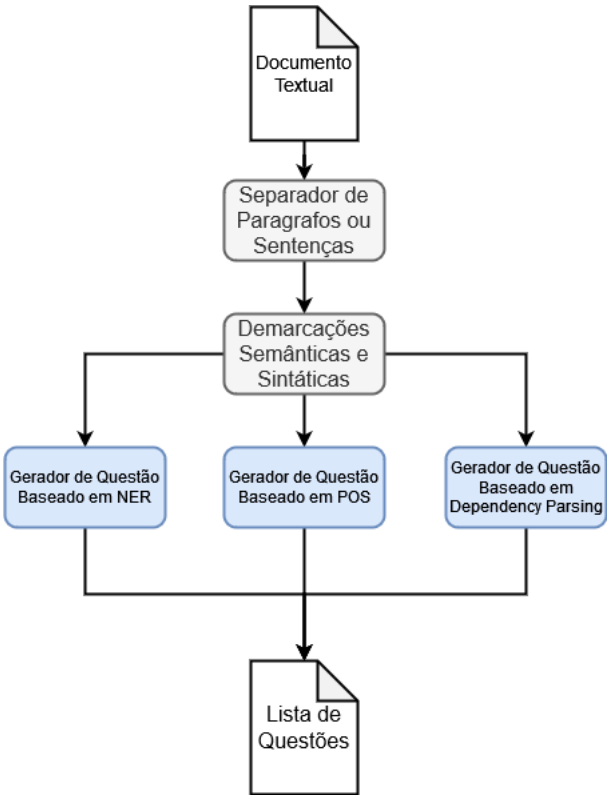
Apesar da predominância da utilização destes demarcadores de palavras de acordo com sua importância na frase (12, 13) houve, também, a presença de adicionais como simplificadores de sentença (14) e remoção de palavras vazias (15, 16). Alem destas, outras técnicas foram analisadas neste contexto, estando entre elas a extração de mapas conceituais com os papéis semânticos do texto (17), identificação de palavras conectivas (18), modelos de questões (19), expressões regulares (20), criação de lacunas em frases para serem preenchidas (21), seletores de palavras chaves no texto (22), aplicação de regras semânticas para tornar afirmações em questões (23) e até mesmo aplicações de regressões lineares nas questões (24).

Técnicas Semânticas e Sintáticas apresentam bons resultados usando algoritmos simples, porém mais complexos de se implementar pela necessidade de conhecimento na área de linguística. Além disso, possuem diversas falhas relacionadas à necessidade de implementação de regras específicas e definição de modelos para a geração das questões.

2.3 Técnicas Semânticas e Sintáticas Apoiadas por I.A.

Já neste paradigma, a utilização de classificação semântica e sintática é fortalecida pela presença de modelos de aprendizado de máquina treinados para reduzir o trabalho ne-

Figura 5 – Exemplo de Arquitetura Utilizando Técnicas Semânticas e Sintáticas



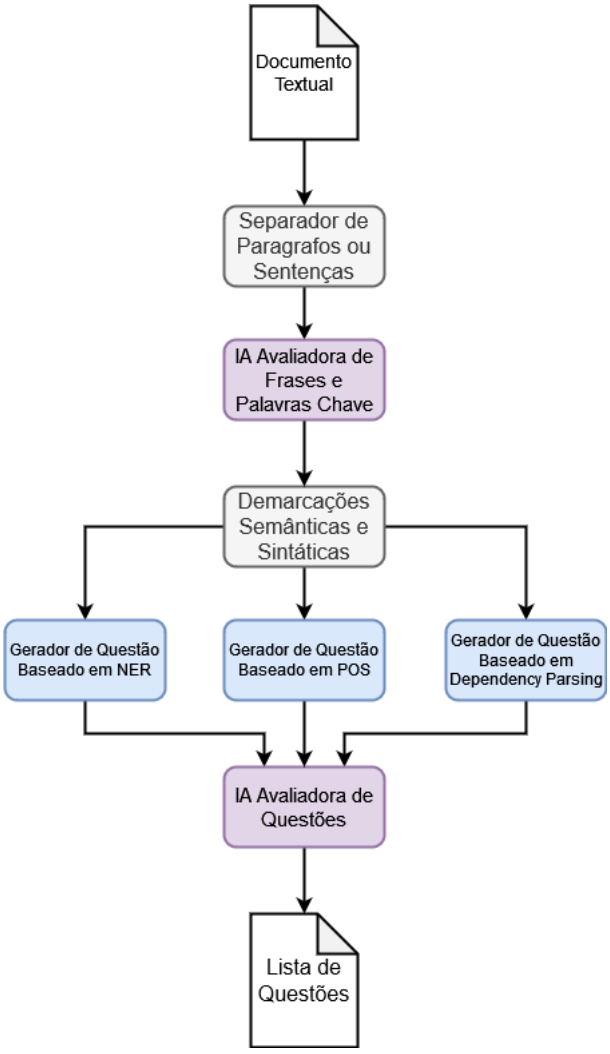
(Gerada pelo autor)

cessário ou melhorar os resultados obtidos pelos algoritmos baseados em regras. Foi observado neste tipo de técnica a predominância em arquiteturas semelhantes às apresentadas na Figura 6, em que os modelos de IA seriam utilizados em duas tarefas principais, sendo estas a classificação de frases e palavras chaves e a classificação das questões geradas, auxiliando os algoritmos geradores de questões e permitindo uma diminuição na necessidade do conhecimento de linguística.

No caso da utilização de modelos para a classificação de frases e palavras chaves, o objetivo do modelo de IA presente se dava na realização de uma pré-filtragem do conteúdo a ser demarcado, diminuindo o trabalho computacional necessário para se demarcar as frases e permitindo a definição de frases e palavras mais significativas para a geração de questão.

Já no caso dos modelos classificadores de questão, o objetivo do modelo é voltado à qualidade das questões geradas. Isso é feito descartando questões que não façam sentido ou realizando um ranqueamento das melhores questões atribuindo uma pontuação às mesmas e selecionando uma quantidade pré determinada de acordo com este ranking. Em trabalhos como os mostrados anteriormente, a geração da questão também dependeu de algoritmos específicos construídos de acordo com a necessidade.

Figura 6 – Exemplo de Arquitetura Utilizando Técnicas Semânticas e Sintáticas Apoiada por IA



(Gerada pelo autor)

Apesar da presença de modelos de IA, não foi possível destacar um tipo predominante de modelo. Foram observados modelos de classificação probabilística como o Naive Bayes (25), que realiza a classificação aplicando o teorema de Bayes no qual se diz possível analisar a probabilidade de um evento baseado no conhecimento das condições extraídas de eventos passados, e até modelos de redes neurais artificiais como LSTM (*Long Short-Term Memory*) (26), uma rede neural profunda amplamente reconhecida por sua capacidade de processamento de dados sequenciais.

Para que tais modelos de IA desempenhassem o papel proposto, diversos datasets foram construídos pelos autores para o treinamento dos mesmos, seguindo então uma abordagem de aprendizado supervisionado. Esta abordagem é amplamente utilizada, principalmente quando se tem a expectativa dos resultados obtidos em relação aos dados

alimentados aos modelos.

Nesta técnica também é possível observar a presença de diversas abordagens que permeiam o paradigma de inteligência artificial, como a utilização de algoritmos de classificação por regressão logística (27), seletores de palavras chaves (28) e agentes inteligentes (29).

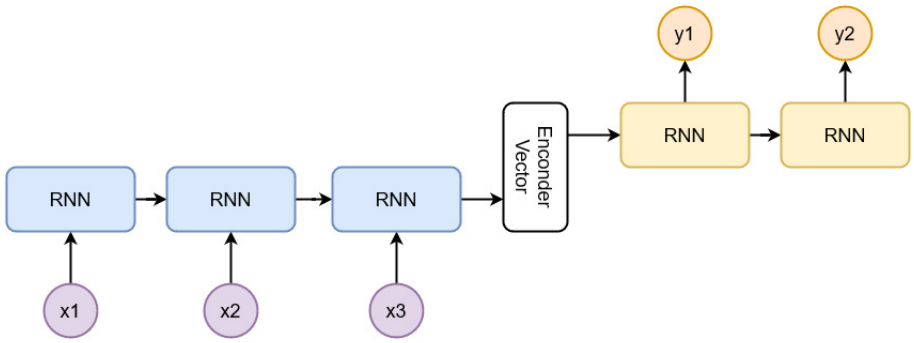
Em relação aos resultados obtidos por estas arquiteturas, não houve um grande avanço comparado aos trabalhos anteriores, porém, através da introdução destes modelos de IA, foi possível identificar uma melhor consistência nos resultados, fazendo com que as questões geradas tivessem maior significância.

2.4 Técnicas de *Deep Learning*

Por fim, a técnica mais utilizada em anos recentes é a de Aprendizado Profundo (*Deep Learning*), propondo redes neurais para geração de questões. Diferente das abordagens anteriores, a utilização de redes neurais profundas possibilitaram a concentração de esforços no desenvolvimento de um único elemento, a rede neural responsável por gerar a questão, excluindo a necessidade de conhecimento em linguística ressaltada anteriormente.

Nesta técnica o processamento geral realizado para a geração da questão está concentrado na rede neural em si, utilizando arquiteturas *encoder-decoder* que são utilizados para realizar o processamento de informações no qual o objetivo da rede é a geração de uma saída sequencial, sendo essa uma sequência de números, uma sequência de bits ou, no caso aplicado, uma sequência de caracteres e palavras que devem representar uma pergunta. O nome deriva da utilização de duas redes neurais separadas denominadas de *encoder* e *decoder*, que são responsáveis por tarefas distintas mas estão encapsuladas no mesmo modelo, como visto na Figura 7.

Figura 7 – Modelo *Encoder-Decoder*



(Adaptado de (30))

Numa arquitetura simples o objetivo do *encoder* é ser o ponto de entrada das informações inseridas na rede, fazendo então um processamento e convertendo estas in-

formações em um vetor de tamanho definido. Seguindo esta linha, o *decoder* então utiliza este vetor para gerar a saída esperada, podendo esta ser de tamanho pré-definido ou variado.

Dentre as redes presentes nas arquiteturas analisadas, seguindo a tendência dos trabalhos que utilizam IA no apoio ao processo de Geração de Questões, foi observada em sua maioria a presença de redes do tipo LSTM, juntamente com redes GRU (*Gated Recurrent Units*). Estas redes possuem como característica comum estarem na categoria de redes neurais recorrentes, redes que foram desenvolvidas no intuito de que a informação processada não se perca ao longo do tempo, ou seja, fazendo com que na geração de cada novo estado, o estado anterior tenha impacto no mesmo. Além disso, ambas possuem um conceito de memória semelhante, no qual a informação prévia que será utilizada no novo estado poderá ser completamente ou parcialmente ignorada através de cálculos definidos de forma diferente em cada um destes dois modelos.

Apesar de estar presente nas abordagens que utilizam IA no apoio aos algoritmos baseados em regra, nestes trabalhos, um outro fator se tornou mais importante, sendo este a tokenização de frases. Como redes neurais podem ser definidas como modelos matemáticos em que as informações processadas e geradas são feitas através de cálculos utilizando números, não é possível fazer o mesmo para palavras ou frases. Para contornar esta limitação a tokenização é utilizada, convertendo cada uma das palavras presentes numa frase em um número que corresponda a mesma. Desta forma o que é processada pela rede neural é uma sequência numérica, sendo esperado a geração de outra sequência numérica. Após o processamento da rede, para que a sequência gerada seja compreendida como um texto ou uma frase, esta passa pelo processo inverso, convertendo cada número na palavra que é representada pelo mesmo antes da apresentação do resultado.

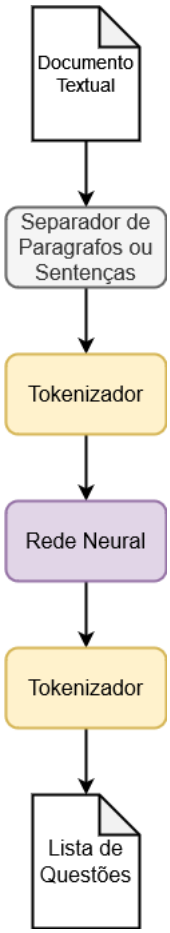
Considerando o maior processamento contido nas redes neurais, uma arquitetura que utiliza tal abordagem se torna mais simples que as demais, porém, concentrado, como exibido na Figura 8.

Diversas abordagens podem ser observadas ao aplicar esta técnica, como a utilização de redes LSTM (31, 32) ou sua variação bidirecional (26) em conjunto com a aplicação de Mecanismos de Atenção (33, 34), Mecanismo de Cópia (35, 36), Reprocessamento das Questões Geradas (37), utilização de mais de um modelo (38), extração de frases chave (39) e o Apoio de Demarcadores Sintáticos (40, 41) já apresentados na Sessão 2.2, além da aplicação de arquiteturas como *encoder-decoder* (42, 43) e arquiterturas de redes neurais adversarias (44).

Também está presente a utilização de redes do modelo GRU (45, 46) ou sua variação bidirecional (47) com o auxílio de Mecanismos de Cópia (48, 49) e Reprocessamento da questão gerada (50)

Outras abordagens podem incluir a utilização de redes CNN (51, 52) e Máquinas de Boltzman (53) na geração de questão.

Figura 8 – Exemplo de Arquitetura Utilizando Técnicas de *Deep Learning*



(Gerada pelo autor)

Além destas, a abordagem que se mostrou com resultados mais sólidos faz a utilização de redes do tipo Transformer (54, 55, 56), possuindo a aplicação de técnicas como a utilização de palavras chave (57), extração de resposta (58), utilização de mais de um tipo de rede para apoiar a geração (59, 60) e também utilizando Demarcadores Sintáticos (61).

Por fim, por possuírem etapas que envolvem aprendizado supervisionado, existe a necessidade da presença de um conjunto de dados de entrada e saída esperada, a fim de que tais redes aprendam como devem realizar a tarefa proposta.

Nos trabalhos analisados foi possível constatar a utilização unânime do dataset SQuAD proposto por Rajpurkar, Jia e Liang (62), que contém mais de 80 mil registros de textos informativos, questões relacionadas a este e a resposta da questão. Este dataset tem sua origem voltada para a utilização em trabalhos que buscam realizar tarefas em torno da tarefa de *question answering*, sendo composto por um texto, uma questão e a resposta a esta pergunta. Por possuir estes elementos é possível adapta-lo para a utilização em tarefas de geração de questões, invertendo o papel da questão com a resposta, sendo

definido a questão como saída esperada da rede.

Os trabalhos utilizando técnicas de *Deep Learning* foram os que possuíram uma maior flexibilidade em seus resultados devido a capacidade da rede de generalizar o conhecimento, podendo assim, realizar a tarefa em diversos domínios diferentes, respeitando apenas a necessidade de se possuir entradas textuais.

2.5 Considerações Finais

Com este capítulo é possível afirmar que geração de questões é uma tarefa que pode ser aplicada a diversos cenários e vem se tornando cada vez mais relevante, desafiando os sistemas computacionais para um novo nível de compreensão de linguagem.

Analisando as técnicas propostas é possível desenhar uma linha do tempo, passando entre os paradigmas que envolviam a utilização de regras fechadas ou continham grande conhecimento de domínio por parte dos autores, com resultados mais simplórios, para técnicas em que este conhecimento de domínio se tornam menos necessários e com aumento na qualidade do resultado final.

As técnicas de *Deep Learning* têm sido cada vez mais amplamente adotadas, especialmente em trabalhos recentes, devido ao seu papel fundamental no avanço significativo da capacidade dos computadores em compreender a linguagem humana de forma mais abrangente.

Com o avanço da tecnologia, técnicas mais robustas tendem a surgir no futuro, permitindo se alcançar melhores resultados nas técnicas apresentadas ou o surgimento de novos paradigmas que permitam o avanço em busca do mesmo resultado.

3 *Transformers*

Como visto no capítulo 2, redes neurais do tipo *encoder-decoder* foram amplamente utilizadas na tarefa de Geração de Questões, obtendo resultados relevantes. Uma grande contribuição neste segmento foi feita por Vaswani et al. (7) ao introduzir as redes do tipo Transformer, um modelo de rede utilizando *encoder* e *decoder* baseado em atenção. No momento de sua publicação, a arquitetura Transformer se tornou o estado da arte nas tarefas de tradução automática, alcançando pontuações melhores que seus antecessores, mesmo com a utilização de menos recursos em seu treinamento.

Com o surgimento de novas tecnologias para o processamento de linguagem natural, houve também avanço neste paradigma, trazendo novas possibilidades e melhores resultados aos desafios já existentes no paradigma.

Nas redes neurais tradicionais os cálculos são realizados por três componentes principais, sendo estes a camada de entrada, em que o dado será inserido, a camada oculta em que estão as diversas camadas que farão o processamento e a camada de saída em que a informação é agrupada e a saída é liberada. No caso dos *transformers*, o modelo é composto por diversos mecanismos separados em uma arquitetura *encoder-decoder* como a descrita anteriormente no capítulo 2. Estes mecanismos e seu funcionamento juntamente com uma descrição da arquitetura são apresentados a seguir.

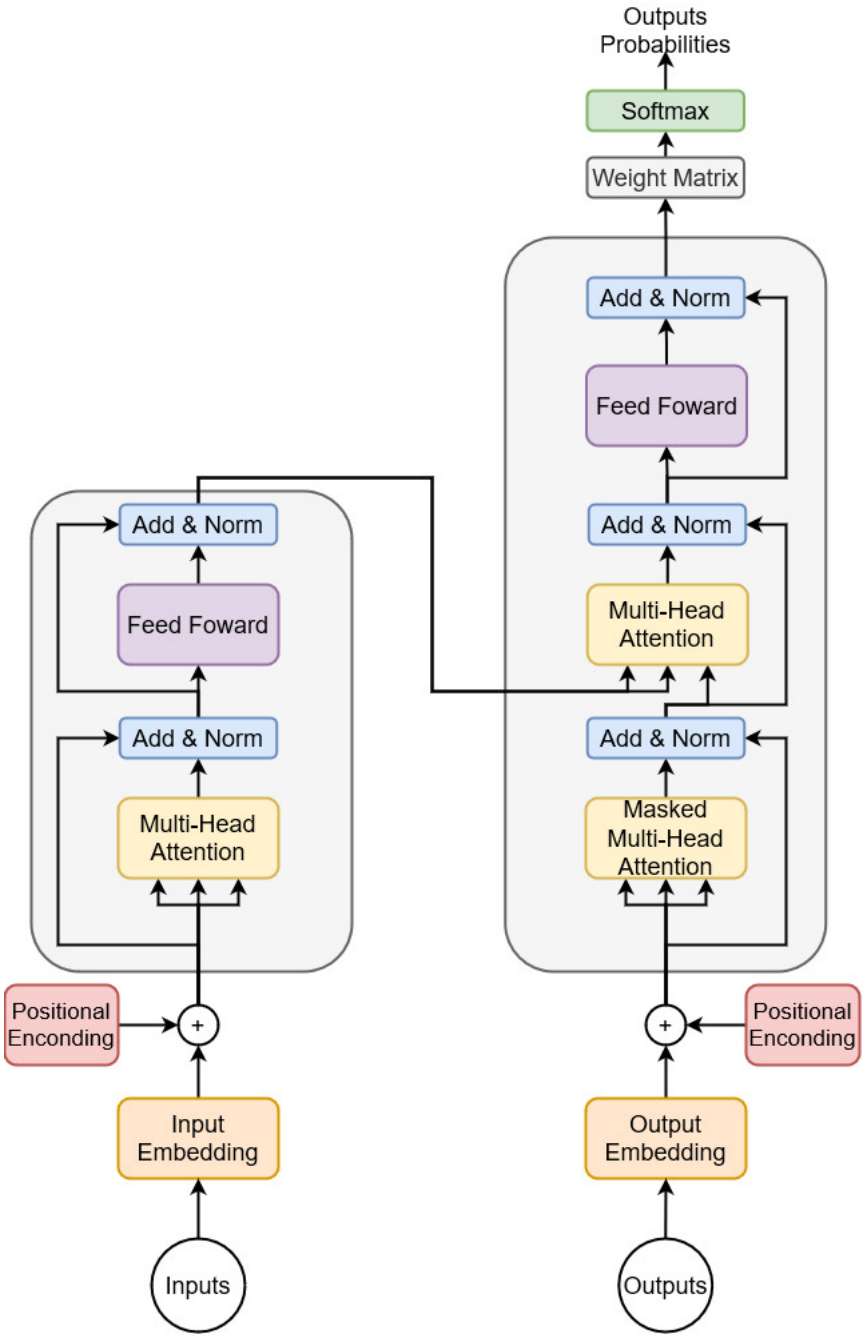
Neste capítulo apresenta-se o funcionamento deste tipo de rede além de explorar as diferentes configurações, que criaram um novo horizonte para o processamento de linguagem natural.

3.1 Arquitetura

Um Transformer, seguindo a arquitetura *encoder-decoder*, é composto por dois blocos, observadas na Figura 9, e que diferem uma da outra em sua composição. O *Encoder*, que recebe a entrada, é composto por duas partes. A primeira é uma aplicação da função *Multi-Head Attention*, que após os cálculos realizados, resulta numa soma e normalização de valores com o valor inicial de entrada. A segunda parte é uma rede *Feed Forward* simples, ou seja, uma rede neural em que a informação é passada através da rede em uma única direção, sendo composta por 3 camadas: a de *input*, a camada oculta e a camada de *output*. Esta rede irá processar os dados e gerará outra soma e normalização com o resultado obtido anteriormente. A matriz resultante deste bloco é disponibilizada para o bloco *Decoder*.

O *Decoder* é composto por três camadas. A primeira camada recebe a saída gerada pela própria rede, calculando a probabilidade da próxima saída através da aplicação da função

Figura 9 – Arquitetura de Redes *transformers*



(Adaptado de (7))

de *Multi-Head Attention* utilizando uma máscara opcional. Essa máscara é utilizada para que a função ignore a parte da matriz que ainda não foi adequadamente gerada. De forma didática, pode-se entender que o terceiro valor da sequência é gerado levando em consideração apenas os valores gerados até o momento, ou seja, o primeiro e o segundo valores. Ao final de cada camada, é realizado um processo de soma e normalização com o resultado anterior, seguindo o mesmo padrão que ocorre nas camadas do *Encoder*.

Para a segunda camada é então utilizado a matriz de valores gerados pelo *Encoder* e a matriz resultante da camada anterior, sendo divididos nas variáveis utilizadas pela função de *multi-head Attention*, em que K e V são os valores do *Decoder* e a variável Q recebe o valor da camada anterior.

A última camada é composta de uma rede *Feed Forward* que disponibiliza os resultados e, após a soma e normalização, passa por uma matriz de pesos e por uma função softmax, uma função que calcula a possibilidade de cada palavra, resultando numa matriz de possibilidades que ditam o próximo valor da sequência a ser gerado.

As matrizes de peso e as redes *Feed Forward* contidas neste modelo, juntamente com as matrizes de peso das funções de *Multi-Head Attention* são as variáveis que são treinadas pelo otimizador a fim de se obter o resultado esperado.

3.2 Cálculos de Atenção

A principal inovação contida neste modelo se dá na utilização de cálculos de atenção, se referindo a uma forma de se distribuir uma pontuação, chamada de *attention score*, na informação inserida. Este tipo de mecanismo pode ser adaptado para diversos tipos de abordagens e tem sido amplamente explorado pela comunidade científica.

O modelo Transformer original surgiu com o intuito de realizar tarefas voltadas a *machine translation*, ou seja, traduzir textos de forma automatizada. Nesta tarefa, o principal problema encontrado pelas redes anteriores era a forma como a tradução das palavras era feita, ou seja, para que uma tradução fosse feita corretamente, haveria a necessidade de se observar certas partes desta frase para cada palavra traduzida.

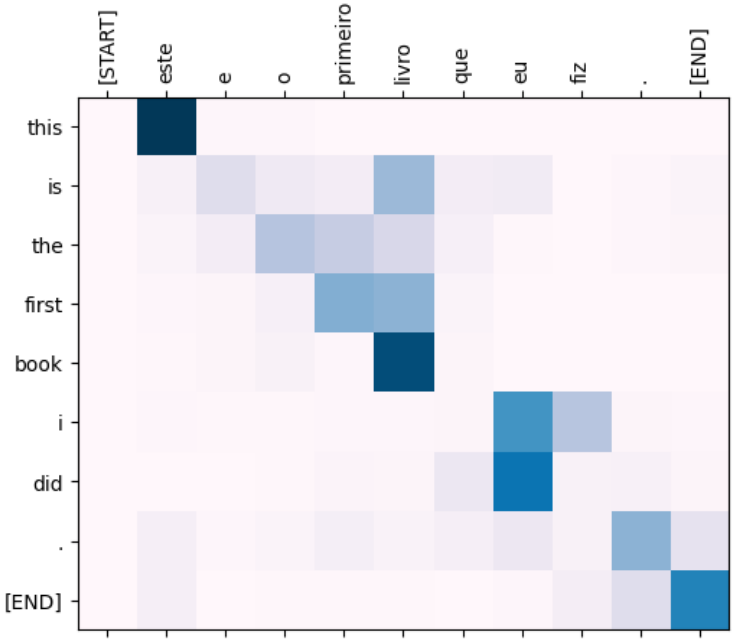
Este problema foi solucionado utilizando este mecanismo de atenção que correlaciona cada palavra da frase original com as palavras da tradução, sendo possível representar o resultado numa matriz, como na Figura 10, onde vemos os valores resultantes do calculo de atenção em forma de mapa de calor.

Observando a palavra "livro", vemos que esta possui correlação muito forte com a palavra "*book*", já que é sua tradução direta, mas também possui correlação com a palavra "*first*" e uma leve correlação com "*the*" . Essas 3 palavras juntos compõe uma sentença que esta totalmente ligada a palavra "livro" a ser traduzida ("*the first book*") e por isso possuem alto valor resultante do calculo de atenção, fazendo com que a rede leve estas palavras em consideração no momento de aplicar a tradução.

3.3 Entrada de dados

Quando a informação é inserida, na forma de uma sequência após ser tokenizada, passa por uma camada denominada de *embedding*. Nesta camada cada valor que representa

Figura 10 – Mapa de Calor da Atenção



(Gerada pelo autor)

uma palavra é convertida em um vetor de tamanho definido de acordo com o modelo, convertendo o vetor de palavras em uma matriz. O intuito desta camada é realizar uma aproximação das palavras que representem características similares em que, por exemplo, a palavra “casa” e “apartamento” teriam valores semelhantes, diferente da palavra “banana” que não possui nenhuma relação às demais.

Com isto o efeito de palavras semelhantes é parecido nos cálculos realizados pela rede, permitindo uma assertividade maior na definição dos resultados obtidos.

Após esta etapa, da mesma forma que os modelos de rede neural citados anteriormente agem, se aplica operações para normalização da posição da sequência. Neste caso, é aplicada uma codificação posicional na forma das equações 3.1 e 3.2. Os resultados são adicionados ao vetor de valores correspondentes a cada palavra, no qual *pos* é a posição, i é a dimensão e *dmodel* é o tamanho definido para o modelo.

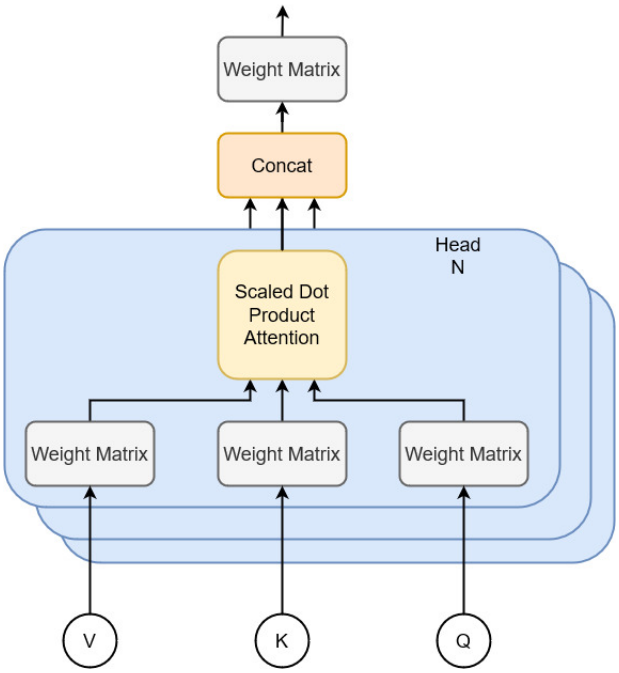
$$PE(pos, 2i) = sen(\frac{pos}{10000^{\frac{2i}{dmodel}}}) \tag{3.1}$$

$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{\frac{2i}{dmodel}}}) \tag{3.2}$$

3.4 *Multi-head Attention*

Outro componente essencial para o funcionamento dos *transformers* é a função conhecida como *multi-head attention*, vista na Figura 11, que aplica de forma paralela as funções de atenção na sequência. Cada cabeça recebe uma cópia da sequência atribuindo o mesmo valor em três vetores diferentes, denominadas de *Query* (Q), *Value* (V) e *Key* (K), e aplica a estes valores uma matriz de pesos correspondente de cada vetor, e que é única para cada cabeça. Após esta alteração de valores, estes passam pela função conhecida como *Dot-Product Attention*, descrita a seguir.

Figura 11 – Função *Multi-Head Attention*

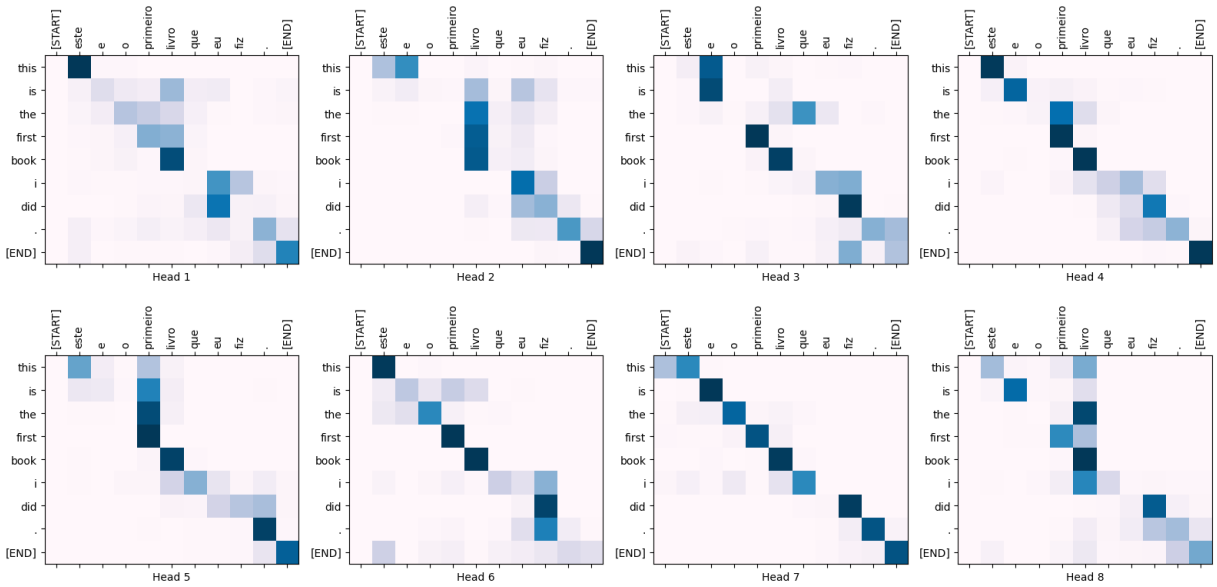


(Adaptado de (7))

Por fim os resultados de cada cabeça são concatenados e inseridos em uma última matriz de pesos. Cada matriz de pesos contida nesta função deve ser alterada de acordo com o treinamento realizado, gerando então o aprendizado esperado.

Desta forma, como no exemplo utilizado na aplicação do *attention score*, na tradução da frase “este é o primeiro livro que eu fiz”, presente na Figura 12, é possível observar a divisão desta pontuação em cada uma das *heads* que realizam os cálculos e enxergar a correlação de cada palavra gerada à sua palavra correspondente.

Figura 12 – Mapa de Calor de Atenção espalhado por oito *Heads*



(Gerada pelo autor)

3.5 *Scaled Dot-product Attention*

A função de *Scaled Dot-product Attention*, que pode ser traduzido de maneira arbitraria para Produto Escalar de Atenção Escalonada, apresentada na Figura 13 é executada de forma que os vetores Q e K são multiplicados na forma de um produto escalar, resultando na correlação entre as duas matrizes. A matriz resultante é então redimensionada, fazendo com que os valores sejam normalizados e estejam dentro do mesmo patamar.

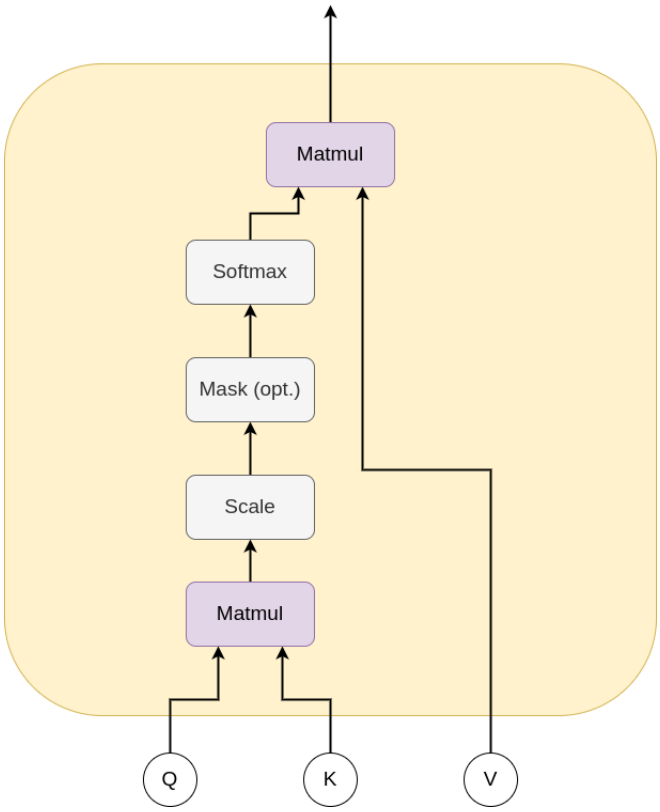
Nesta função há a opção de se aplicar uma máscara, que é uma matriz indicativa de quais valores da matriz devem ser utilizados ou não, tendo sua utilidade destacada na descrição da arquitetura do Transformer.

Este valor passa pela função softmax, convertendo essa matriz em uma matriz de números entre zero e um. Por fim, a variável V é multiplicada da mesma forma, utilizando produto escalar, com a matriz resultante.

3.6 Treinamento e Resultados

Como o foco desta rede foi sua utilização na tarefa de *machine translation*, foi-se definido pelos autores como datasets de treinamento na tradução entre textos da língua inglesa para o alemão com o dataset WMT 2014 English-German, composto de cerca de 4.5 milhões de pares de sentenças nas respectivas línguas e 37 mil palavras únicas, e na tradução de textos da língua inglesa para o francês com o dataset WMT 2014 English-to-French, composto de 36 milhões de pares de sentenças além de 32 mil palavras únicas.

Figura 13 – Função *Scaled Dot-Product Attention*



(Adaptado de (7))

Para se treinar redes profundas, o hardware sempre se tratou de um ponto importante, pelo fato de que tais redes, por serem de tamanho maior que as redes tradicionais, demandam de muito recurso computacional para tal.

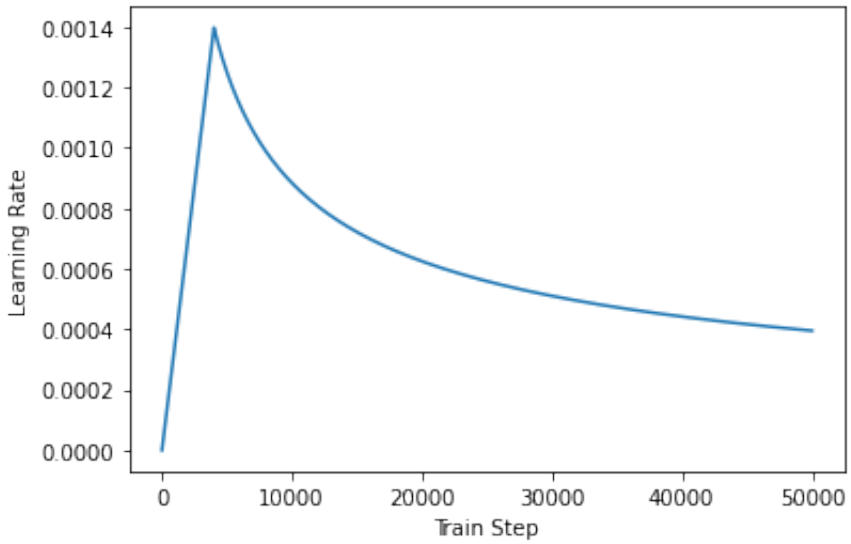
O trabalho original utilizou oito placas gráficas NVidia do tipo P100, que possui 12 GB de memória e poder computacional de até 21.2 Teraflops cada, fazendo com que a rede demorasse entre 12 e 84 horas de treinamento para se adaptar aos datasets propostos, ou o equivalente entre 100 mil e 300 mil passos de treinamento.

Outro ponto importante no treinamento é o otimizador utilizado, que é o componente responsável por calcular as medidas de erro da rede e causar as mudanças necessárias nas variáveis para que este erro seja reduzido, ou seja, gerando o aprendizado na rede. O otimizador Adam (63) utilizado neste trabalho permite a utilização de parâmetros ajustáveis, sendo estes α, $\beta 1$, $\beta 2$ e ε, em que α, ou *learning rate*, é o parâmetro principal deste algoritmo, que dita para o cálculo de erro e aprendizado, quão forte será a mudança realizada nos parâmetros na rede e portanto, foi desenvolvido uma função específica para a alteração deste ao longo do tempo de treinamento, apresentada na equação 3.3.

$$LearningRate = \sqrt{dmodel} \times min(\sqrt{stepNum}, stepNum \times warmupSteps^{-1.5}) \quad (3.3)$$

Desta forma a dimensão do modelo, representado pela variável *dmodel*, impacta diretamente no tamanho da correção realizada sendo multiplicado pelo menor resultado das duas equações seguintes, com o cálculo baseado na quantidade de passos dados durante o treinamento, representado pela variável *stepNum* e outro cálculo utilizando juntamente a quantidade de passos de aquecimento, na variável *warmupSteps*, que resulta num aumento gradual do valor resultante até um patamar em que começa a diminuir até estabilizar, como mostrado na Figura 14.

Figura 14 – Learning Rate



(Gerada pelo autor)

Com isto temos a utilização de 0,9 em $\beta 1$, 0,98 para $\beta 2$ e 10^{-9} para ε. Já o valor de α é determinado pela equação 3.3 utilizando 4000 para *warmupSteps* e os demais valores sendo variados de acordo com a composição do modelo e o passo de treinamento no momento.

O trabalho contou com a definição e treinamento de diversos modelos com tamanhos diferentes, a fim de se encontrar os melhores resultados possíveis. Com isto foram definidos principalmente dois modelos, conhecidos como Base e Big, em que o modelo Base utilizou 512 para a dimensão, 2048 para a quantidade de camadas *Feed Forward* e oito *heads* para os cálculos de atenção, enquanto o modelo Big utilizou o dobro desses valores nas dimensões definidas.

Aplicando tais valores e tempos de treinamento, o modelo Big alcançou resultados melhores que os apresentados até o momento, se tornando estado da arte para *machine translation* nos idiomas especificados, além de possuir um custo computacional aproximado muito menor, destacando-se principalmente em relação ao trabalho de Gehring et al. (64) que era o estado da arte anterior, empregando $1,2 \times 10^{21}$ teraflops de custo computacional, enquanto foi possível obter resultados melhores com $2,3 \times 10^{19}$ teraflops, cerca de 50 vezes menos.

Para comparar os resultados o algoritmo BLEU foi utilizado, que é capaz de definir um valor variando entre 0 e 100 para a qualidade das traduções realizadas ao comparalas diretamente com os resultados esperados. Os melhores resultados no momento da publicação para tradução inglês-alemão e inglês-francês eram de 26,36 e 41,29, respectivamente. O modelo Big foi capaz de atingir o resultado de 28,4 em tradução inglês-alemão, uma diferença de 2,04 pontos, e 41,8 para tradução inglês-francês, uma diferença de 0,51.

3.7 Avanços no modelo

Após a apresentação destes resultados a arquitetura transformer ganhou a atenção de diversos grupos de pesquisa, dando início a uma série de trabalhos que buscaram adaptar tal rede para outras finalidades ou aperfeiçoar a arquitetura a fim de se obter melhores resultados em tarefas específicas ou generalistas.

Uma grande contribuição foi feita por Radford et al. (65), que implementou uma nova forma de treinamento para os *transformers*. Estes modelos, por tratarem de processamento de linguagem natural, dependem de uma grande quantidade de dados préprocessados para o treinamento supervisionado. Para amenizar tal necessidade, foi proposto o treinamento em duas etapas juntamente com um novo modelo utilizando este treinamento, o GPT (*Generative Pre-trained Transformer*), onde existe a presença de apenas blocos decoders.

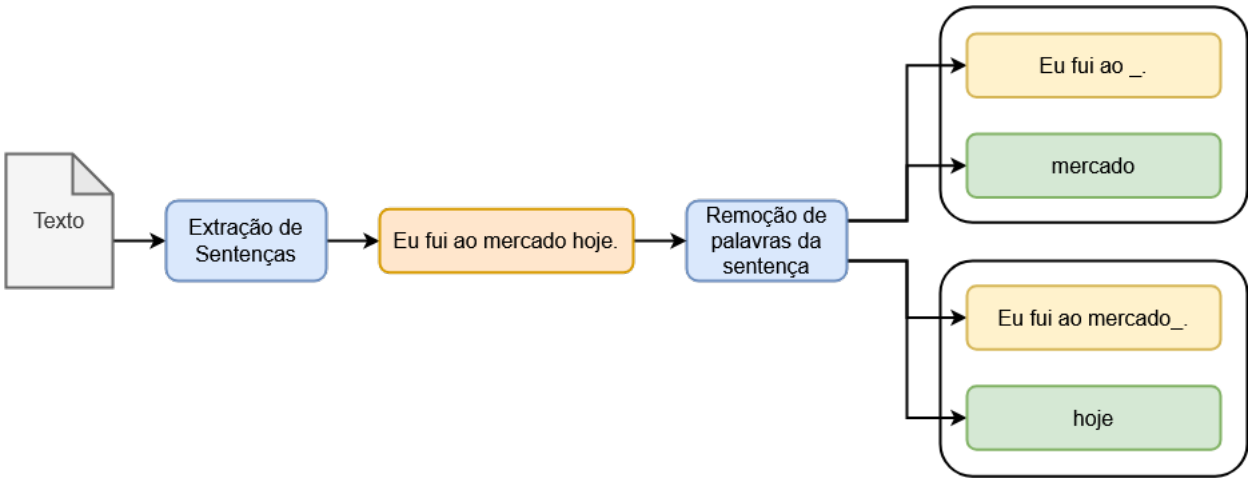
A primeira etapa está concentrada em aplicar técnicas de treinamento semi-supervisionado, em que os textos disponibilizados são, de forma automatizada, convertidos em grupos de sentenças e, como visto na Figura 15, disponibilizados com o final da sentença faltando. O treinamento ocorre então através de uma tarefa de preenchimento de próxima palavra, tendo como objetivo adivinhar a palavra correta. O intuito deste tipo de treinamento está na transferência do conhecimento de linguística para a rede, fazendo com que esta compreenda como a linguagem em questão é estruturada.

Já a segunda etapa tem como foco realizar o treinamento específico da rede para a tarefa proposta, que pode variar de acordo com o objetivo da implementação. Nesta etapa o treinamento supervisionado é utilizado da mesma forma que nos trabalhos iniciais, tendo como objetivo reduzir a distância entre o resultado esperado e o resultado obtido.

Uma das principais vantagens desta abordagem é, além de se poder utilizar mais informação no treinamento, disponibilizar modelos pré-treinados que podem ser ajustados de acordo com a necessidade e permitindo a distribuição dos mesmos em repositórios públicos.

Com a adoção deste método para o treinamento de *transformers*, outro avanço foi realizado por Devlin et al. (66) ao introduzir o modelo BERT, acrônimo de *Bidirectional Encoder Representation from transformers*, tendo uma arquitetura Transformer em que o

Figura 15 – Exemplo de conjunto de Treinamento Semi-Supervisionado



(Gerada pelo autor)

decoder não está presente mas os blocos de *encoder* são agrupadas em maiores números.

Como o nome sugere, este modelo implementa bidirecionalidade ao Transformer, permitindo ao modelo analisar a sequência presente na sentença da esquerda para a direita e da direita para a esquerda, possibilitando um melhor aprendizado nas relações das palavras ao longo da frase.

Outra inovação contida no modelo BERT é sua forma de treinamento, que contou não apenas com o treinamento semi-supervisionado visto em (65), mas também com outro tipo de treinamento semi-supervisionado, no qual o objetivo da rede é adivinhar o próximo *token* presente na sequência a partir de um número limitado de palavras.

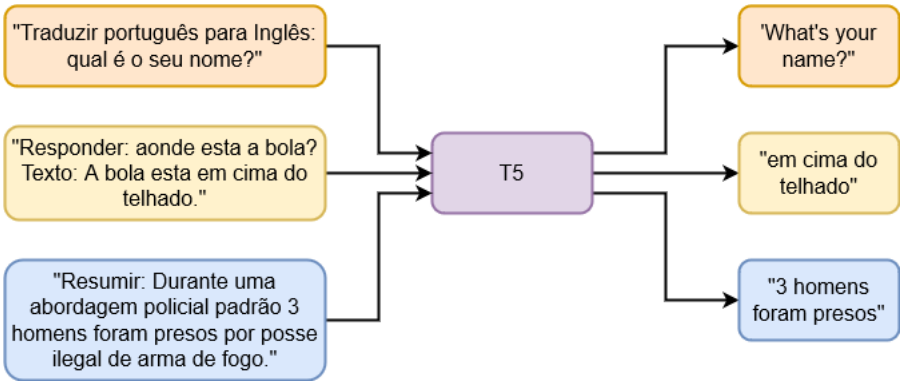
Este modelo também se tornou estado da arte para diversos benchmarks envolvendo processamento de linguagem natural, como *General Language Understanding Evaluation*, ou GLUE (67), e *question answering* utilizando o dataset SQuAD v1.1 e 2.0 (68).

Por fim, outra mudança de paradigma ocorreu com o trabalho de Raffel et al. (69). Anteriormente, cada modelo de Transformer era treinado em uma única tarefa e este trabalho buscou explorar outra abordagem, analisando a quantidade de conhecimento que poderia ser armazenado em uma única rede. Com isto o modelo T5 (*Text-to-Text Transfer Transformer*) foi introduzido, tendo sua implementação bem parecida com a do Transformer original, porém com foco na geração de textos e uma forma de tratamento diferente.

O treinamento realizado apresentou traços parecidos aos trabalhos citados anteriormente, utilizando esse novo paradigma de treinamento semi-supervisionado. No entanto, este divergiu na etapa de *fine-tuning*: para cada tarefa, foi inserido um prefixo que a identifica, como mostrado na Figura 16. Após a inserção do prefixo, o modelo foi ajustado ou adaptado especificamente para cada tarefa, permitindo que ele aprendesse a realizar a

tarefa de interesse de maneira mais eficiente e precisa.

Figura 16 – Modelo de Treinamento T5



(Gerada pelo autor)

Assim, o início de cada sequência contém uma espécie de comando, em que o objetivo da rede alteraria diretamente relacionado com este, podendo assim ser treinado em diversas tarefas utilizando um único modelo e a lista de comandos referentes às tarefas treinadas.

O modelo T5 também obteve resultados que se tornaram o novo estado da arte em diversas tarefas, superando o modelo BERT no dataset SQuAD e no benchmark GLUE.

3.8 Considerações Finais

Conforme evidenciado neste contexto, os *transfomers* têm emergido como a vanguarda no campo de pesquisa em uma ampla variedade de tarefas relacionadas à compreensão e ao processamento de linguagem. Dessa forma, esses modelos têm sido considerados um fator preponderante para fomentar a exploração de abordagens envolvendo esse domínio específico. Além disso, os avanços contínuos nesse modelo em particular têm servido como uma manifestação palpável do entusiasmo compartilhado pela comunidade científica, que está empenhada em ampliar ainda mais o alcance e a aplicabilidade dessa arquitetura inovadora.

4 Geração de Questões no Cenário de Sistemas Tutores Inteligentes

Neste capitulo é abordado inicialmente o modelo de aprendizagem proposto através do sistema tutor em desenvolvimento, em que cada aluno pode ser classificado em um de quatro grupos, e, então, é apresentado como este trabalho se relaciona a estes grupos e como se propõe a resolver os desafios atrelados a esta abordagem no que diz respeito a preparação de atividades para os grupos que se beneficiam destas.

4.1 Modelos de Aprendizagem

Dentre várias pesquisas que consideram os estilos de aprendizagem relevantes no ambiente educacional, David Kolb (4) realça a importância da experiência prática para a aquisição de novos conhecimentos, passando por um ciclo de experiência, reflexão, conceptualização e aplicação.

Neste ciclo, a experiência pode ser descrita como qualquer evento externo, como atividades em grupo ou testes aplicados, ou interno, como reflexões e sentimentos. A reflexão sobre estas experiências leva à compreensão do que ocorreu e do porquê, construindo a etapa de conceitualização, na qual são formadas teorias e ideias sobre o que foi apresentado. Por fim, a aplicação é a utilização deste conhecimento adquirido em novas situações.

Assim, se propõe que as pessoas podem ser classificadas de acordo com suas preferências sobre como o conteúdo é apresentado a elas e como elas reagem a esse conteúdo. Em sua proposta original Kolb criou uma taxonomia com quatro estilos de aprendizagem, sendo estes:

- Adaptadores ou Acomodadores, que acham mais fácil aprender usando a metodologia de tentativa e erro, aprendendo com a experimentação, observando resultados do que implementam, geralmente por sua intuição, na resolução de problemas.

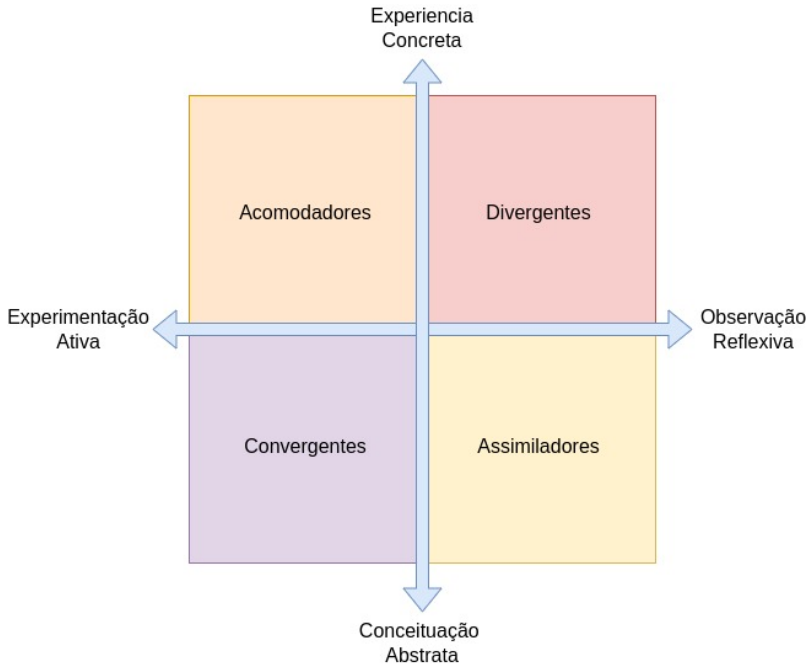
- Assimiladores, que preferem aprender pela compreensão dos conteúdos apresentados, que devem ser claros e formais, permitindo a formulação de modelos analíticos que possam ser explorados.

- Divergentes, que tendem a ter melhor desempenho em atividades em grupo, com melhor percepção das sensações e observação do ambiente para extrair conhecimento, possuindo mente aberta e criativa.

- Convergentes, que aprendem melhor aplicando conteúdos previamente apresentados para resolver problemas reais, levando a teoria para a prática.

Na Figura 17 é possível ver como esses quatro estilos se relacionam. Nela se observa que eles são distribuídos em quatro quadrantes, que dividem os estilos quanto ao seu comportamento (ativo ou reflexivo) e quanto ao formato da informação (teórico ou prático). Por exemplo, temos os acomodadores ocupando o quadrante em que as pessoas são ativas (experimentação ativa) e que aprendem melhor fazendo do que vendo alguém fazer ou explicar (experiência concreta).

Figura 17 – Modelo de Kolb



(Adaptado de (70))

O processo de ensino convencional não trata pessoas com diferentes estilos, tratando todos de modo uniforme. Entretanto, existe uma corrente de pensamento pedagógico (71)(72) que considera que ensinar apenas para um grupo de perfil tende a fazer com que alguns alunos se destaquem, enquanto outros podem ser marginalizados ou até mesmo reprovados naquele curso.

O uso de uma abordagem que considere essas diferenças resulta em um aprendizado mais eficaz para a maioria dos alunos, se não todos. No entanto, ensinar em estilos diferentes, utilizando uma abordagem mais individualizada, é inviável sem o auxílio de algum ferramental didático, como os sistemas tutores.

4.2 Objetivo

Procurou-se deixar claro nos capítulos anteriores a necessidade da implementação de métodos que permitam a automação da geração de conteúdos no ambiente de sistemas tutores inteligentes. Reforçou-se também as diferentes abordagens disponíveis, além de apresentar as tecnologias mais avançadas utilizadas no processamento de linguagem natural. Esses aspectos levam à proposta descrita neste capítulo, iniciando-se com uma conceituação mais detalhada do cenário em que se insere e finalizando por uma descrição da proposta aqui feita.

Este projeto trata a utilização das tecnologias levantadas para o processamento de linguagem natural, especificamente na tarefa de Geração de Questões, tendo como objetivo elevar o nível de autonomia dos sistemas tutores inteligentes. Na maioria dos casos, um sistema tutor inteligente é desenvolvido com uma coletânea de atividades finitas, que serão utilizadas ao longo do processo de aprendizagem e avaliação. Com o objetivo de se individualizar o ensino, a utilização de grupos de aprendizado pode ser uma abordagem promissora, mas, é necessário fornecer diferentes tipos de tarefas ou fontes informacionais para cada um dos grupos, o que pode tornar a tarefa um tanto impraticável para a aplicação no mundo real. Isso ocorre pois para cada grupo e para cada tópico a ser ensinado, se deve realizar a criação de conteúdos e formatos diferentes.

Uma das formas de se facilitar este processo está na automação da geração do conteúdo, o que pode ser abordado em diversas etapas do processo de ensino, desde a disponibilização da informação em resumo de textos previamente apresentados, como também a automação da geração de avaliações. No âmbito deste projeto está a geração de atividades que possam testar e avançar o nível de compreensão do aluno para cada tópico apresentado, sem a necessidade de um professor ativamente realizando a criação destas.

Portanto, chega-se à utilização de mecanismos de Geração de Questões a fim de se criar, a partir de origens textuais, atividades aos grupos de ensino que, em seu ciclo de aprendizado, tenham preferências pela resolução de problemas. Em particular, dentro da taxonomia de Kolb, temos que os acomodadores, que possuem mais facilidade de aprendizagem com a experimentação, e os convergentes, que assimilam o conteúdo de forma mais eficaz em problemas práticos, são os grupos que mais se beneficiam de formas automatizadas de se gerar questões.

Entre as diferentes abordagens disponíveis, as abordagens que utilizam algoritmos baseados em regras são descartados pois a definição das regras é um processo custoso e demorado. Isto pode tornar o processo de desenvolvimento de uma ferramenta usando essa abordagem uma tarefa complexa e que muitas vezes não possibilitará uma utilização real da mesma, principalmente pela dificuldade em se adaptar ao cenário proposto.

Observando as abordagens que utilizam redes neurais na geração das questões, é possível perceber uma melhor adaptabilidade das redes para conteúdos variados. Isto

condiz com a proposta destes algoritmos, que é aprender como a tarefa é realizada e replicá-la em qualquer outro cenário. Com estes algoritmos, treinados com diversos exemplos de questões e diversos tópicos de ensino diferentes, não existe a barreira do domínio, ou seja, não existe a limitação da aplicação da técnica em apenas um tópico específico, como biologia ou redes de computadores, por exemplo.

Além disto, as frequentes quebras de recordes que estes algoritmos vem obtendo permite indicar que são os melhores disponíveis para processamento de linguagem natural, tendo sua adaptabilidade possível para a tarefa proposta. Portanto a utilização de *transformers* para esta tarefa se mostra a melhor abordagem, permitindo a utilização de algoritmos já treinados e disponibilizados por autores diferentes a fim de se encontrar o melhor resultado possível.

Com isto, busca-se desenvolver um algoritmo que utiliza inteligência artificial para, a partir de origens textuais, gerar a questões e respostas para a utilização no ensino através de um sistema tutor inteligente baseado em grupos de aprendizado.

4.3 Metodologia

Os estudos apresentados nos capítulos anteriores indicaram uma melhor performance e sofisticação nos algoritmos que utilizam Redes Neurais em sua construção, mais especificamente as do tipo Transformer. Estas redes são atualmente as mais citadas nas tarefas de *Natural Language Understanding*, sendo as responsáveis pelo avanço do estado da arte em diversos tópicos, fazendo com que sejam uma boa escolha para este projeto de forma clara. Tais redes possuem diversas arquiteturas e podem ser encontradas em diversas formas, como nos repositórios dos autores, em que são disponibilizadas a fim de se gerar uma prova para as afirmações contidas em seus trabalhos, ou em bibliotecas especializadas na disponibilização das mesmas visando o uso simplificado destas tecnologias.

Entre estas bibliotecas destaca-se a biblioteca *transformers* (73), que é implementada pela empresa HuggingFace. Nesta, diversos pesquisadores têm a possibilidade de publicarem suas arquiteturas e permitirem a utilização simples e descomplicada para a comunidade, além de fornecerem redes com etapas de treinamento já concluídas. Devido ao grande poder computacional necessário para o treinamento destas redes explanado de maneira mais detalhada a frente, este trabalho busca utilizar arquiteturas que já possuem algum treinamento na atividade proposta, selecionando entre elas a que possui melhor desempenho comprovado, de acordo com a arquitetura da solução proposta.

Esta seleção é auxiliada com o uso de algoritmos para pontuar a qualidade dos resultados produzidos na geração de questões. Exemplos desses algoritmos podem ser encontrados nos algoritmos BLEU, METEOR e ROUGE. O uso desses algoritmos serve, também, para verificar as comparações apresentadas pelos autores das diversas arquiteturas para

implementação dos transformers encontradas na literatura.

O Algoritmo BLEU(*Bilingual Evaluation Understudy*) é uma forma de avaliação automatizada desenvolvida por Papineni et al. (74), sendo proposta inicialmente para tarefas de *Machine Translation*, com o intuito de metrificar a qualidade das traduções geradas, comparando-as com uma ou mais traduções de referência, gerando uma pontuação resultante. O algoritmo pode ser ajustado para considerar conjuntos de palavras dentro da frase, variando entre grupos de um à quatro (BLEU-1 a BLEU-4), fazendo com que no grau um, apenas palavras sejam comparadas, ou seja, apenas verificando se a palavra está aparecendo na frase gerada, já no grau quatro, grupos de quatro palavras são comparadas, fazendo com que a ordem que estas palavras apareçam na frase seja relevante. Apesar do intuito inicial deste algoritmo, este é bastante utilizado em outras tarefas que envolvam a geração de texto e, por isso, é utilizado tanto para métricas de Geração de Questões e *Question Answering*.

De forma semelhante, o algoritmo METEOR (*Metric for Evaluation of Translation with Explicit ORdering*) proposto por Banerjee e Lavie (75) também foi desenvolvido para a verificação de traduções e é uma forma de se comparar textos gerados com um resultado esperado, mas, diferentemente do BLEU, nao possui diferentes graus, mas principalmente, é mais sensível à ordem das palavras na tradução, levando em conta não apenas as palavras individuais, mas também as frases e a estrutura gramatical.

Já o algoritmo ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*) proposto por Lin (76), se originou com o intuito de permitir a avaliação da qualidade de resumos gerados por sistemas de sumarização, sendo mais utilizado em sua versão L (ROUGE-L), em que se é comparado a sequência mais longa de palavras que aparecem em ordem comum tanto na referência quanto no resumo gerado. de forma semelhante aos outros algoritmos existe a necessidade de se comparar com uma saída esperada, levando em consideração tanto a precisão quanto a fluência geral do resumo gerado, utilizando um conjunto de regras de correspondência de palavras para avaliar a similaridade entre o resumo gerado e a referência. Também similarmente aos algoritmos anteriores, este é utilizado nas demais tarefas envolvendo geração de texto.

Para verificar as arquiteturas descritas a seguir, o algoritmo principal para geração das questões foi implementado em Python. Após a implementação, a fase de testes buscou identificar se os resultados esperados foram atingidos, utilizando textos pré-selecionados. Observe-se que a implementação contém comentários e documentação para que seja possível realizar manutenções futuras por outros pesquisadores que sigam esta linha de pesquisa, com o intuito de ser utilizado em um Sistema Tutor Inteligente completo.

4.3.1 Arquitetura da Solução

Analisando os modelos que obtiveram os melhores resultados nos *benchmarks* previamente citados, foi observado que a utilização não apenas do contexto da pergunta, ou seja, da fonte informacional, mas também da resposta esperada influenciou diretamente nestes resultados. Logo, para que a rede possa formular a questão corretamente, esta também utiliza a resposta a esta questão. Esta abordagem tem sido a principal escolha para diversos trabalhos relacionados. Entretanto, se torna uma barreira entre a pontuação obtida nos *benchmarks* apresentados e sua utilização em ambiente real, em que o intuito é a geração da questão a partir apenas do contexto informacional, sem a resposta esperada.

Para contornar este requisito foi encontrado na literatura a utilização de mais de uma tarefa para a geração da questão, no qual primariamente há a tarefa de buscar a resposta, ou seja, encontrar no texto disponibilizado, um ponto de informação relevante que será considerado como a resposta, sendo este utilizado como a resposta esperada da pergunta a ser gerada.

Sendo assim é possível especificar quatro arquiteturas baseadas nas tecnologias previamente vistas, sendo apresentadas abaixo. Estas arquiteturas foram desenvolvidas com o intuito de abranger uma necessidade do ambiente de Sistemas Tutores Inteligentes que diz respeito a correção da resposta fornecida pelo aluno ao realizar as atividades propostas, fornecendo junto a questão gerada, a resposta esperada que pode ser utilizada por algoritmos posteriores que serão responsáveis pela correção da resposta dada pelo aluno.

4.3.1.1 Arquitetura Multi-Modelo

Na Arquitetura Multi-Modelo, vista na Figura 18, há a presença de dois modelos de redes neurais do tipo Transformer, denominadas Modelo Buscador de Respostas e Modelo Gerador de Questões. O intuito do Modelo Buscador de Respostas é justamente realizar esta busca inicial por uma informação relevante, que será utilizada como resposta esperada, utilizando o texto fornecido, separado por parágrafos, como entrada do algoritmo.

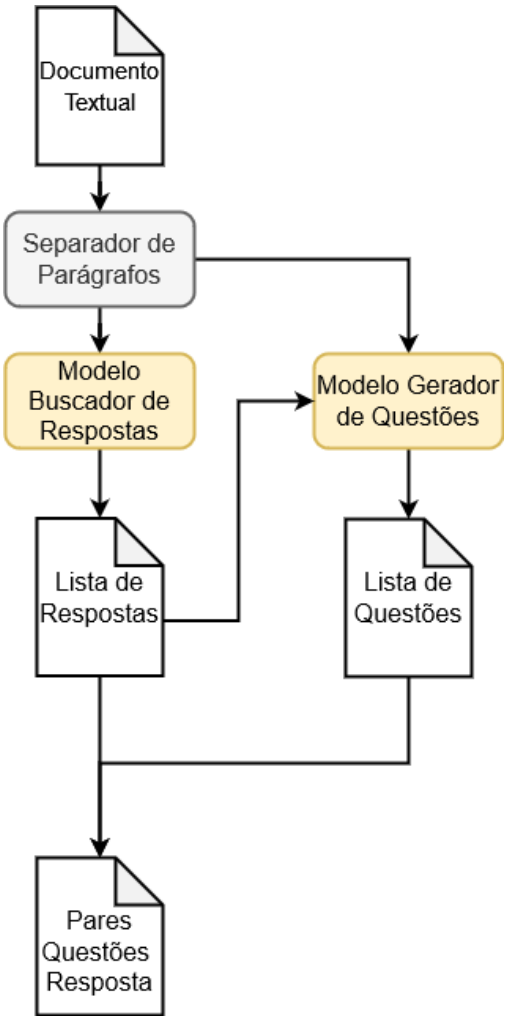
O passo seguinte é executado pelo Modelo Gerador de Questões que, agrupando as respostas extraídas pelo modelo anterior aos parágrafos utilizados para gerar tal resposta, realizará a geração das questões.

Por fim, o algoritmo deve agrupar as questões geradas juntamente com as respostas utilizadas para gerar as questões, formando uma lista de questões e respostas que poderá ser utilizada pelo Sistema Tutor Inteligente.

4.3.1.2 Arquitetura de Modelo Único

Já na Arquitetura de Modelo Único, vista na Figura 19, busca-se, utilizando paradigmas previamente descritos, utilizar apenas uma rede neural do tipo Transformer, treinada nas duas tarefas.

Figura 18 – Arquitetura Multi-Modelo



(Gerada pelo autor)

Com isto há a necessidade de executar o processamento em duas etapas, de forma semelhante à arquitetura anterior, porém usando a mesma rede. Tal abordagem permite a obtenção da lista de questões e respostas, mas, reduzindo o tamanho em disco necessário, pois exige a presença de apenas uma rede.

Busca-se observar nesta arquitetura qual o verdadeiro impacto gerado quando se treina uma rede em mais de uma tarefa e mensurar se este impacto é significativo o suficiente para confirmar a necessidade da utilização de dois modelos diferentes, sendo cada um treinado em cada uma das tarefas utilizadas.

4.3.1.3 Arquiteturas com Modelo Gerador de Respostas

A adição de um modelo treinado em responder perguntas tem como objetivo melhorar o resultado final da arquitetura, incrementando a qualidade das respostas que serão ligadas às questões geradas. A geração de novas respostas ocorre mesmo considerando que

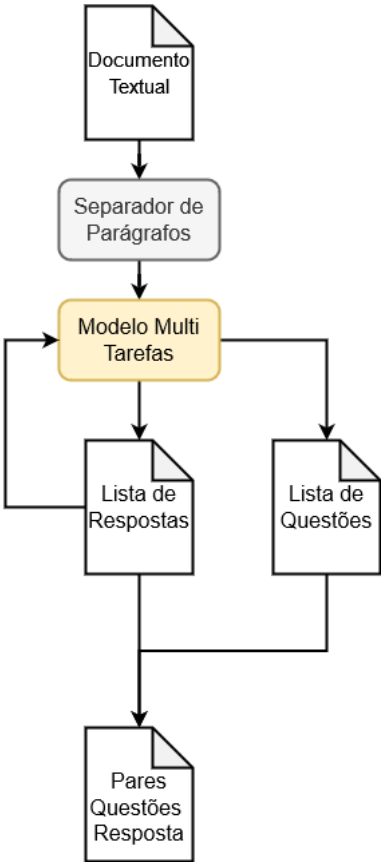


Figura 19 – Arquitetura de Modelo Único

(Gerada pelo autor)

as questões foram geradas a partir de respostas previamente definidas, criando um duplo refinamento no par questão-resposta ao analisar o texto novamente no intuito de se responder tal pergunta.

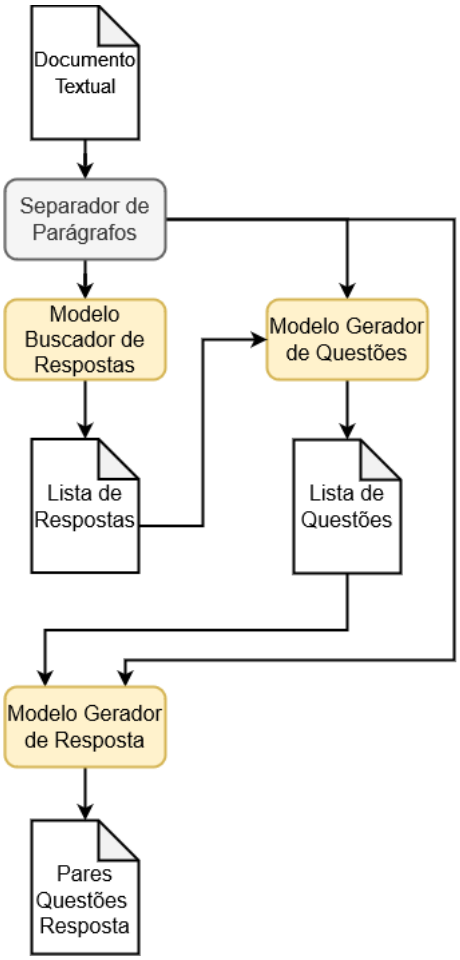
Sendo assim, é possível incrementar as arquiteturas Modelo Único e Multi-Modelo, convertendo-as em Modelo Único com Gerador de Resposta, apresentada na Figura 20a, e Multi-Modelo com Gerador de Resposta, apresentada na Figura 20b.

Estas arquiteturas seguem os mesmos passos das duas arquiteturas anteriores no processo de geração de questões. A diferença fica por conta de um passo adicional que é o fornecimento desta questão ao Modelo Gerador de Resposta, juntamente com o parágrafo original utilizado para se extrair a informação e gerar a questão.

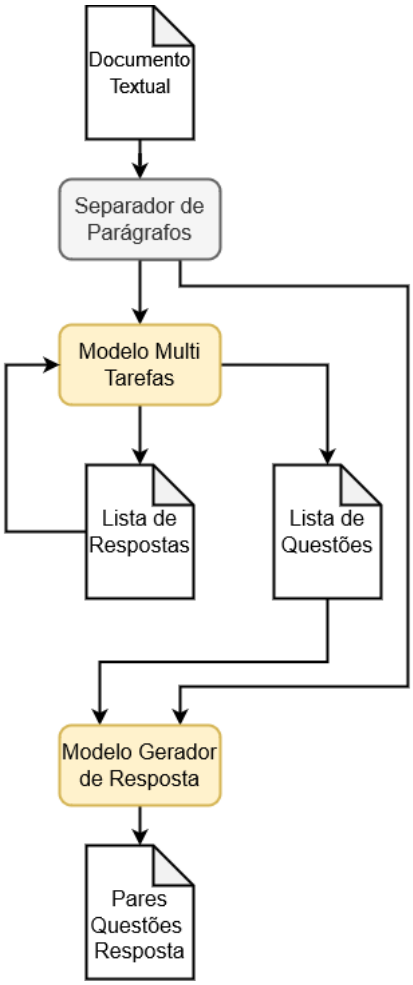
A adição do novo Modelo Gerador de Respostas é responder à pergunta gerada, agora com mais informação, para que a nova resposta seja então utilizada como resposta esperada do aluno dentro da lista de questões e respostas fornecidas ao Sistema Tutor Inteligente.

Figura 20 – Arquiteturas com Gerador de Resposta

(a) Arquitetura de Modelo Único



(b) Arquitetura Multi-Modelo



(Gerada pelo autor)

4.4 Considerações Finais

Do exposto neste capítulo é possível identificar que a utilização de redes *transformers* para a tarefa de Geração de Questões no ambiente de Sistemas Tutores Inteligentes é possível e pode ser feita sem grandes complicações. As arquiteturas e metodologias apresentadas demonstram que a proposta do trabalho é factível e interessante, pois possibilita a geração automatizada de atividades para aprofundar o aprendizado dos alunos.

5 Validação e Testes

Como apresentado no capítulo anterior, uma abordagem para se gerar questões no ambiente de sistemas tutores inteligentes seria a utilização de uma das arquiteturas propostas. Para validar os resultados e definir qual arquitetura seria a melhor para a aplicação no STI em desenvolvimento, foram realizados testes que são apresentados neste capítulo juntamente com os resultados.

5.1 Ambiente de Desenvolvimento

Para o desenvolvimento da solução se utilizou um ambiente de desenvolvimento em nuvem denominado Google Collaboratory, também conhecido como Colab, onde, utilizando um estilo de demarcação conhecido como *notebook*, permite a programação e inserção de textos e imagens para o auxílio na documentação do código. Outro ponto importante é seu foco no desenvolvimento de soluções que possuam algum nível de utilização de *machine learning* ou outras técnicas de inteligência artificial, disponibilizando máquinas virtuais preparadas para este tipo de solução.

Com esta ferramenta foi possível executar, testar e validar todas as técnicas apresentadas neste trabalho. Além disto existe também uma facilidade no compartilhamento dos códigos desenvolvidos, visto que este ambiente possui diversas ferramentas de compartilhamento permitindo uma fácil disseminação.

O desenvolvimento foi realizado na linguagem de programação Python em sua versão 3.7, sendo adicionadas algumas bibliotecas (PyTorch e *transformers*) especializadas na construção de redes neurais.

A biblioteca Torch é um *framework* de aprendizado de máquina de código aberto, escrita na linguagem Lua, que fornece ferramentas para construir, treinar e implantar modelos de inteligência artificial. Ela é amplamente utilizada em pesquisas que envolvam a utilização de IA e pode ser aplicada em diversas tarefas, como processamento de linguagem natural, visão computacional e processamento de sinais. Além disso, ela é baseada em tensores, oferecendo recursos avançados, como operações matemáticas e estatísticas específicas para aprendizado de máquina, módulos para criar modelos de redes neurais e técnicas de otimização para treiná-los, além de permitir a execução de cálculos em GPUs para acelerar o processo de treinamento. Utiliza-se a versão em Python desta biblioteca, denominada de PyTorch, que possui o mesmo objetivo e foi adaptada para a linguagem utilizada

Já a biblioteca *transformers* é um pacote de software de aprendizado de máquina de código aberto, desenvolvido pela companhia Hugging Face, que fornece um conjunto de

ferramentas para criar, treinar e implantar modelos de linguagem natural pré-treinados e personalizados. Ela é construída em cima da biblioteca PyTorch, a versão em Python da biblioteca Torch descrita anteriormente, e é baseada em arquiteturas de modelos de *transformers*.

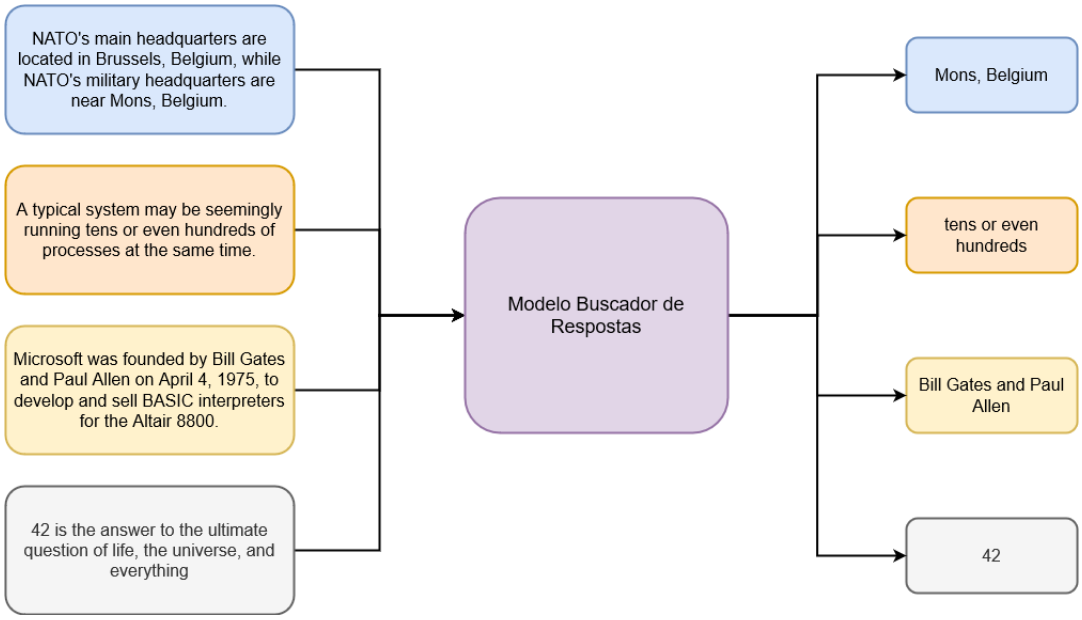
Estes elementos foram as bases utilizadas para a implementação das arquiteturas propostas no capítulo anterior.

5.2 Modelos Utilizados

Para a construção das arquiteturas propostas anteriormente foram utilizadas diversas redes neurais diferentes, todas baseadas no modelo Transformer, com diversos treinamentos diferentes.

Para a etapa de extração de respostas iniciais, sendo representada nas arquiteturas como Modelo Buscador de Respostas, buscou-se na literatura exemplares de redes treinadas nesta tarefa, sendo então definida por fim uma rede do tipo T5, abordada na seção 3.7, realizando processamento como mostrado na Figura 21, a fim de, como descrito anteriormente, fornecer uma resposta que será utilizada na próxima etapa para gerar a questão

Figura 21 – Processamento do Modelo Buscador de Respostas

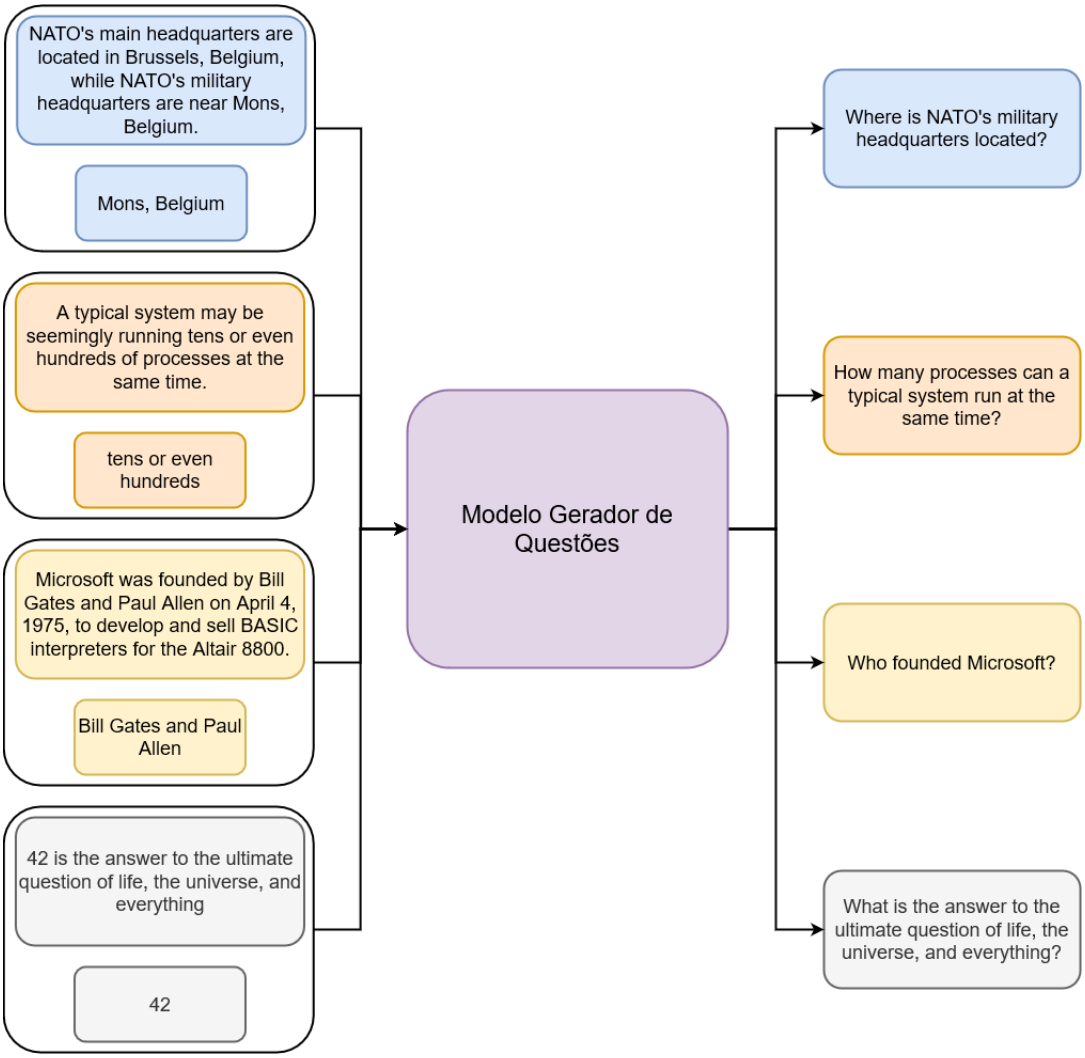


(Gerada pelo autor)

Para a etapa de geração de questões, representada nas arquiteturas como Modelo Gerador de Questões, redes do tipo T5 foram utilizadas, tanto como modelo único multitarefa, como modelo específico de geração de questões ou extração de resposta inicial.

Sendo assim, estas redes realizam o processamento como exibido na Figura 22, gerando assim a questão que será ligada à resposta utilizada e, no caso das arquiteturas em que um modelo Gerador de Resposta está presente, fornecendo o resultado para tal modelo.

Figura 22 – Processamento do Modelo Gerador de Questões



(Gerada pelo autor)

As redes T5 foram utilizadas por sua capacidade de absorver uma grande quantidade de conhecimento, permitindo que estas tenham treinamento em diversas tarefas diferentes e também estejam preparadas para lidar com tarefas complexas, como é o caso da geração de questões.

Outro fator levado em consideração é a pouca disponibilidade de redes treinadas neste tipo de tarefa, sendo, na maior parte dos casos, possível encontrar resultados das redes, mas com pouco ou nenhuma informação em relação ao treinamento realizado para a obtenção destes resultados.

Analisando a tabela 1, é possível observar que esta rede não possui o melhor resultado

obtido no benchmark de Geração de Questões utilizando o dataset SQuAD, mas possui um resultado muito próximo do estado da arte atual, com uma diferença de cerca de quatro pontos no algoritmo BLEU-4 e supera o estado da arte no algoritmo METEOR com uma pequena diferença. Estas pontuações foram definidas através da aplicação dos algoritmos com o mesmo nome, que resulta em pontuações variando em 0 e 100, sendo melhor descrito na seção 5.3.

Tabela 1 – Pontuações em benchmark das redes analisadas

Modelo	Autor	Pontuações		
		BLEU-4	METEOR	ROUGE-L
ERNIE-GEN Large	(77)	25.40	26.92	52.84
T5 QG-HL Base (utilizada)	(78)	21.32	27.09	43.60
BERT-HLSQG	(79)	20.33	23.88	48.23
PLQG	(33)	16.38	20.25	44.48
NQG-RC	(58)	11.86	16.28	39.37

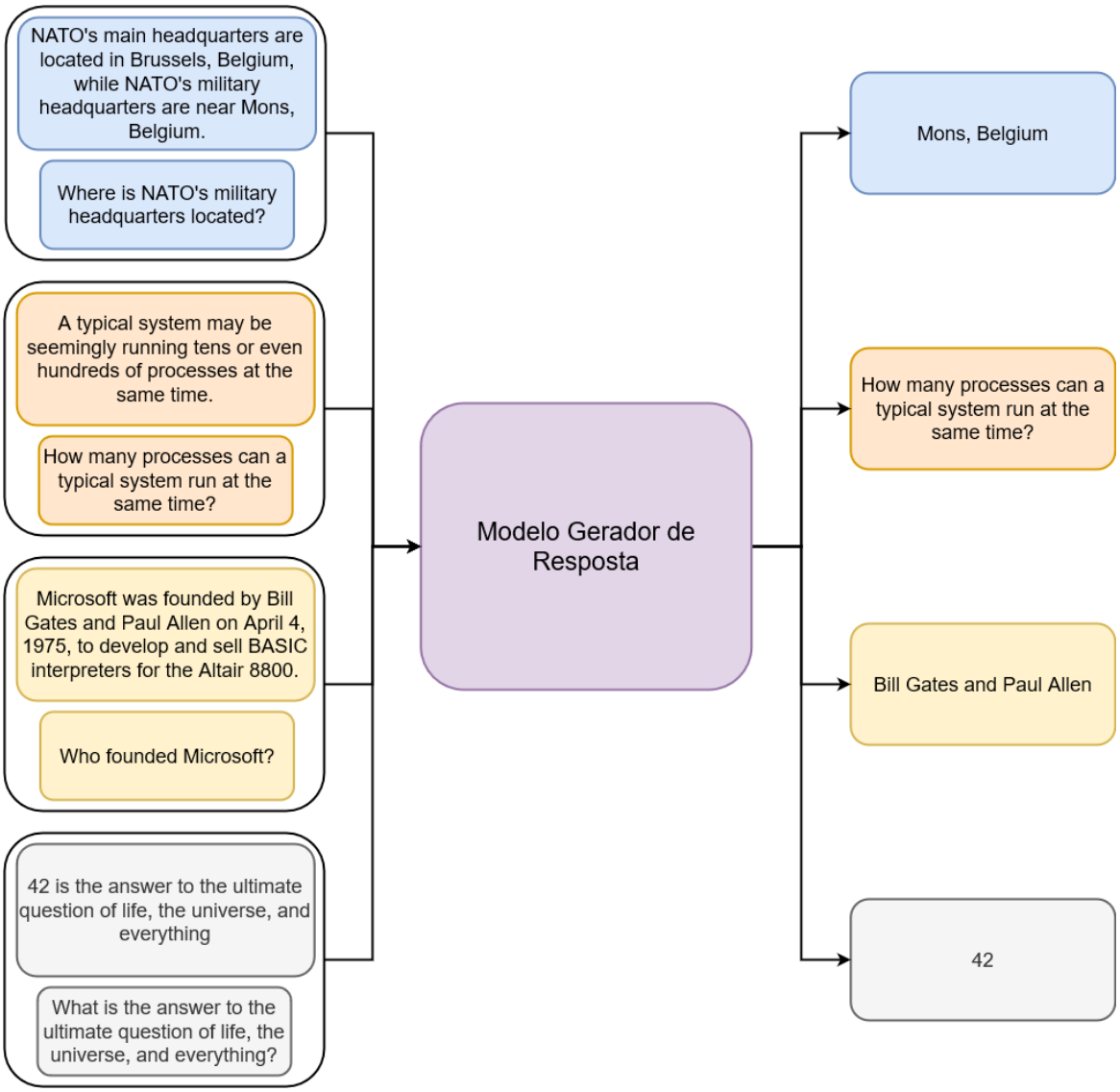
Esta pequena diferença aliada a grande necessidade de poder computacional, como apresentado pelo autor (77), no qual foram necessárias diversas GPUs do tipo Nvidia Tesla V100 além de sete dias de treinamento contínuo, tornaram a realização do treinamento desta rede para sua utilização na arquitetura desinteressante, sendo então optado pela utilização da melhor rede disponibilizada com o treinamento realizado (78).

Por fim, para a etapa de geração de resposta final foram selecionadas alguns modelos de redes treinadas nesta tarefa, a fim de se realizar uma comparação dos resultados obtidos. Como ressaltado anteriormente, a tarefa de *question answering* é muito mais disseminada na comunidade científica, além de já ter atingido uma maturidade muito maior que a Geração de Questões.

Por esse motivo, as redes analisadas possuíam uma pontuação semelhante nos resultados de benchmarks demonstrados pelos autores, sendo assim realizada uma bateria de testes utilizando as questões geradas pelas arquiteturas para se fazer uma comparação de resultados, chegando então a definição da rede DistilBERT (80) como Modelo Gerador de Resposta. Este modelo é baseado na rede Bert original, mas possui a vantagem de ser mais leve e mais rápido que seu antecessor, sem perder o poder de processamento e a qualidade dos resultados obtidos. Os resultados obtidos pela rede, seguindo os exemplos anteriores, é apresentada na Figura 23.

Com os modelos expostas, as arquiteturas apresentadas anteriormente foram construídas a fim de se comparar seus resultados e definir a melhor abordagem para o cenário proposto.

Figura 23 – Processamento do Modelo Gerador de Respostas



(Gerada pelo autor)

5.3 Formas de Avaliação

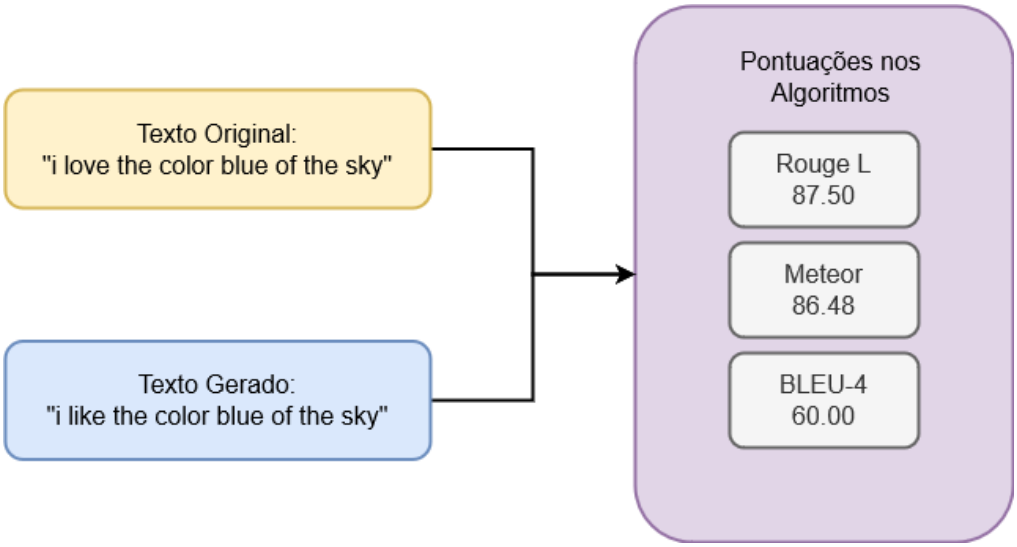
Para que fosse possível medir a qualidade dos modelos treinados, alguns métodos de avaliação automatizada foram utilizadas pela literatura, sendo estes os algoritmos BLEU(74), METEOR(75) e ROUGE(76), buscando assim uma forma de classificar seus resultados e permitindo que tabelas comparativas como a 1 fosse construída. No entanto, essas formas automatizadas de avaliação se mostraram um desafio devido à complexidade da tarefa, e por isso, os autores citados previamente buscaram em algoritmos de outras tarefas uma forma de se comprovar a qualidade de seus trabalhos.

Todas estas formas de metrificação, amplamente utilizadas na comparação de tarefas em que o objetivo é a geração de algum texto definido, não funcionam como deveriam para

esta tarefa. Nas Figuras 24, 25 e 26 é possível identificar como o resultado dos algoritmos ao comparar uma frase simples utilizada de exemplo.

No caso de resultado exato a pontuação atribuída para todos os algoritmos é de 100, e, quando se há uma pequena alteração na frase comparada, a pontuação diminui em todos os algoritmos, demonstrado na Figura 24.

Figura 24 – Pontuações dos algoritmos comparando resultado semelhante



(Gerada pelo autor)

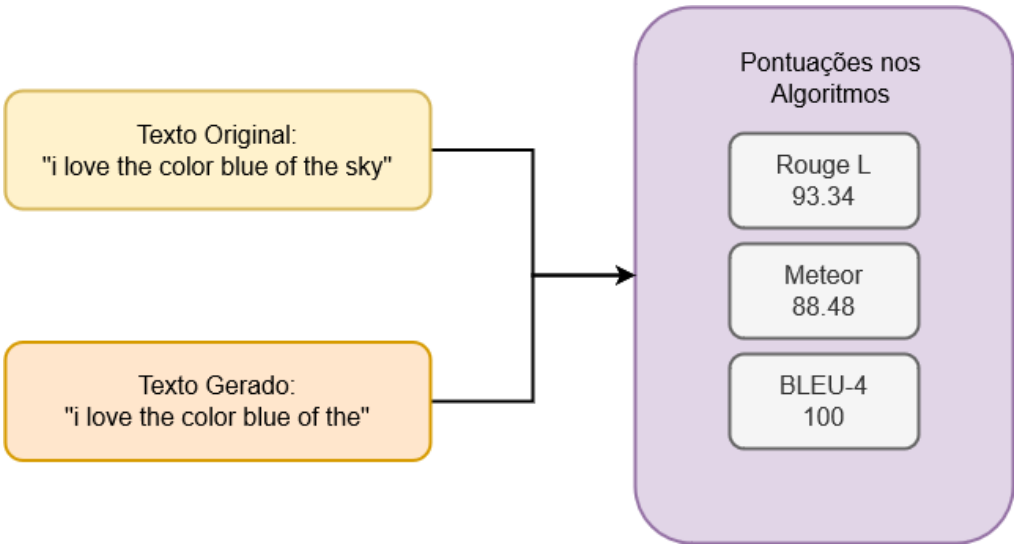
No entanto, quando se é comparada uma frase gerada com uma palavra faltante, como na Figura 25, sua pontuação é maior se comparado com a frase com a palavra trocada. Isso se dá ao fato destes algoritmos utilizarem a comparação das palavras e suas posições para tentar atribuir a pontuação final.

Do mesmo modo, caso a palavra trocada tenha um sentido totalmente diferente, visto na Figura 26, sua pontuação é equivalente a frase com palavra do mesmo sentido devido a forma que estes algoritmos funcionam na comparação de textos gerados, não levando em consideração o sentido da palavra que foi trocada.

Além das questões levantadas nas Figuras acima, outro grande impacto na utilização destes algoritmos está na comparação em si, já que, tais algoritmos dependem de uma resposta esperada para calcular o resultado. Como demonstrado na Figura 27, isto cria um problema quando se é possível gerar mais de uma questão com os dados fornecidos.

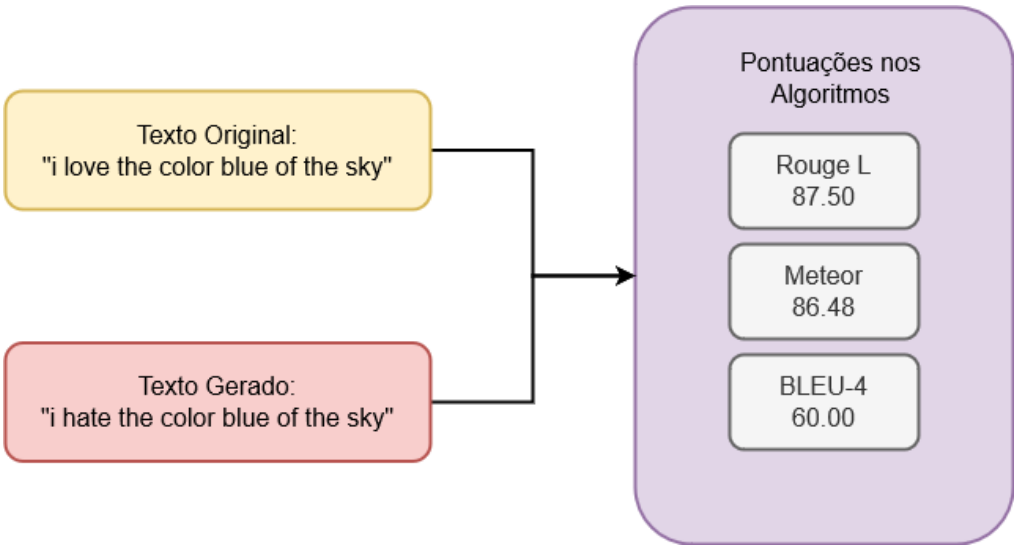
No exemplo utilizado, extraído do próprio dataset SQuAD, no qual a rede utilizada deveria gerar uma questão em que a resposta seria “Von Miller”, um jogador de futebol americano que jogou pelo time de Broncos entre 2011 e 2021, é possível identificar o problema ressaltado. Na fonte informacional é possível identificar algumas informações relacionadas a este jogador, sendo então factível a criação de diversas questões diferentes. No entanto como a comparação só pode ser realizada com a questão original presente no

Figura 25 – Pontuações dos algoritmos comparando resultado com palavra faltante



(Gerada pelo autor)

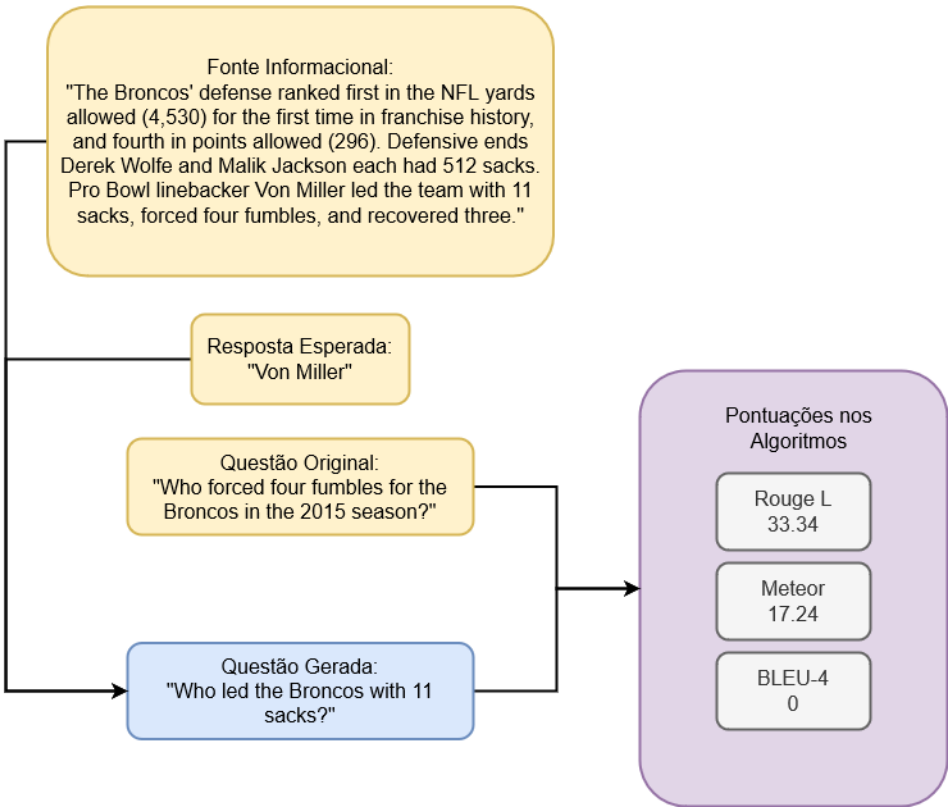
Figura 26 – Pontuações dos algoritmos comparando resultado com palavra de sentido contrário



(Gerada pelo autor)

dataset, isto resulta em uma pontuação extremamente baixa para a questão gerada, apesar desta ser uma questão formada corretamente e ser possível a sua utilização numa aplicação real. Isto mostra como estes algoritmos utilizados na avaliação podem não condizer com a realidade dos resultados obtidos e, seguindo diversos exemplos na literatura, leva este trabalho a realizar, adicionalmente as pontuações já demonstradas, uma etapa de avaliação feita manualmente para julgar a utilização destas questões geradas em situações reais.

Figura 27 – Pontuações dos algoritmos comparando questões diferentes



(Gerada pelo autor)

Portanto, apesar destes algoritmos servirem como uma base inicial, foi realizada uma avaliação qualitativa com critérios abstratos, a fim de se identificar os resultados obtidos pelas arquiteturas. Foram avaliados as questões geradas, de forma a indicar se estas estavam formuladas corretamente, se faziam sentido com o texto e se possuíam uma certa importância no contexto. Da mesma forma foi avaliado as respostas a essas questões, levando em consideração a sua formulação, a presença da informação no texto e como a resposta se encaixava com a pergunta realizada.

5.4 Testes Realizados

Para mensurar e validar os resultados obtidos pelas arquiteturas, foram aplicados diversos testes relacionados à capacidade das redes de produzirem seus resultados, variando a fonte informacional inicial de frases curtas até textos inteiros. Devido a limitação da arquitetura de produzir apenas uma questão por fonte informacional, foi definido então a estratégia de separar os textos em parágrafos, para que, a cada passo de execução, um parágrafo fosse inserido no sistema e uma questão fosse gerada, definindo assim o algoritmo de separação de parágrafos como parte da arquitetura.

Para colher resultados mais abrangentes e verificar a capacidade de aplicação, textos informativos foram selecionados. Inicialmente os testes se deram utilizando um capítulo do livro *Operating Systems: Three Easy Pieces* (81), que trata de tópicos de sistemas operacionais, sendo reconhecido por possuir uma linguagem acessível, além de ser disponibilizado gratuitamente para download. O Capítulo escolhido explana sobre a função e utilização de processos, além da funcionalidade dos sistemas operacionais de realizar várias tarefas ao mesmo tempo. Este capítulo possui o total de 2469 palavras divididas em 33 parágrafos.

Para complementar os testes, mais dois textos foram utilizados, mas, sendo extraídos da Wikipédia, uma plataforma amplamente conhecida por ser uma enciclopédia grátis disponibilizada na Internet, sendo atualizada e revisada por usuários voluntários e que trata de diversos assuntos que podem ser utilizados como ponto inicial para uma compreensão mais profunda sobre o tópico desejado.

O primeiro texto foi extraído da página sobre Algoritmos Genéticos (82), uma forma de algoritmo inspirada na natureza que busca, através dos mecanismos de seleção, reprodução e mutação, selecionar candidatos, ou seja, resultados para o problema, que melhor se encaixem na solução desejada, utilizado de principalmente em problemas de otimização e de busca. Este texto utilizado possui 1281 palavras divididas em 16 parágrafos.

Já o segundo texto trata sobre a empresa Microsoft (83), narrando os acontecimentos históricos, desde sua fundação até a atualidade, e traz informações sobre os principais produtos da empresa, fatos relevantes como as alterações de liderança e também informações sobre questões financeiras como o lançamento na bolsa de valores e informes de lucros. Por fim, o texto extraído possui 2911 palavras divididas em 20 parágrafos.

Estes três textos foram selecionados com o intuito de demonstrar a capacidade das arquiteturas de realizar o processamento intencionado com tópicos e estilos de escrita diferenciadas, sendo este o principal diferencial apresentado pelas técnicas que optaram pela utilização da metodologia proposta.

5.5 Resultados Obtidos

No total, utilizando a estratégia de separação do texto por parágrafo, foram geradas 69 questões, sendo estas 33 para o capítulo sobre processo do livro de sistemas operacionais, 16 do artigo sobre Algoritmos Genéticos, e 20 para o artigo sobre a empresa Microsoft, de acordo com a quantidade de parágrafos de cada texto. O processamento foi realizado para cada arquitetura proposta, sendo possível observar na tabela 2 os resultados obtidos.

Como descrito anteriormente, as questões geradas pelas arquiteturas foram avaliadas de forma qualitativa, sendo atribuídas valores de Ruim, Regular, Boa e Ótima. Questões consideradas ruins foram definidas ao considerar que não faziam muito sentido, tendo

Tabela 2 – Avaliação das Questões e Respostas Geradas por Arquitetura

Arquitetura	Questões				Respostas			
	Ótima	Boa	Regular	Ruim	Ótima	Boa	Regular	Ruim
Modelo Único	15	22	16	16	44	3	8	14
Multi-Modelo	18	27	13	11	52	0	7	10
Modelo Único com Gerador de Resposta	15	22	16	16	48	5	4	12
Multi-Modelo com Gerador de Resposta	18	27	13	11	56	4	1	8

problema em sua formulação, ou então, que ao serem extraídas do contexto do parágrafo, não poderiam ser respondidas. Questões Regulares seriam questões que foram bem formuladas mas não teriam muita relevância ao comparar com as demais questões, ou seja, não teriam muito impacto ao serem aplicadas para identificar a capacidade ou nível de conhecimento do aluno. Questões Boas seriam utilizáveis para estes aspectos, sendo bem formuladas e tendo relevância ao relacionar com o assunto do texto. Já as Ótimas seriam questões que, dado o assunto tratado, melhor representam o conhecimento presente no texto e seriam utilizadas perfeitamente para o propósito do sistema. Na Figura 28 é possível observar exemplares de cada tipo de questão.

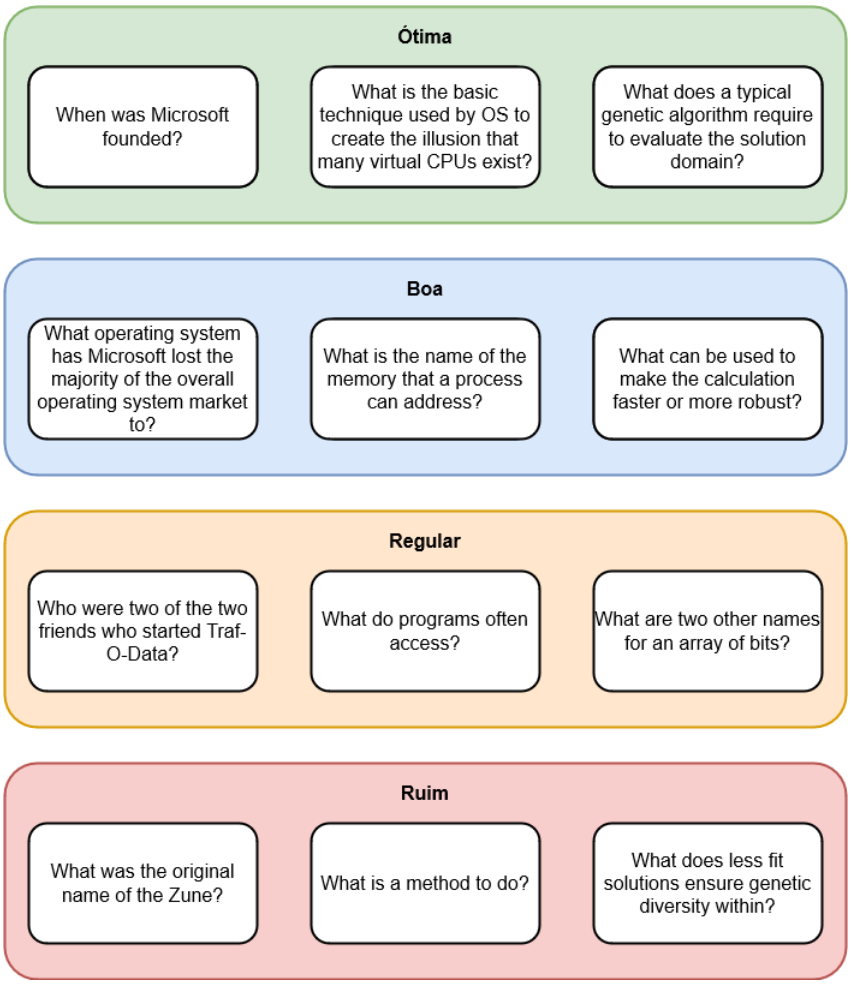
De forma semelhante, as respostas foram avaliadas, seguindo os mesmos valores de avaliação, mas com cada valor representando aspectos diferentes. Respostas Ruins não se encaixavam na pergunta realizada, sendo assim não utilizáveis. Respostas Regulares possuíam relação com a pergunta gerada, mas não se encaixavam totalmente ao resultado esperado. Já as respostas Boas se mostraram relacionadas a pergunta, possuindo uma boa relação com o resultado esperado, mas poderiam ser melhor formuladas. Por fim, as respostas Ótimas completam perfeitamente a questão utilizada e dizem respeito totalmente ao resultado que se espera. Da mesma forma, na Figura 29 é possível observar exemplares de cada tipo de resposta.

É importante ressaltar que, no caso de questões mal formuladas, como é o caso da questão “What is a method to do?”, ou questões com informações incorretas, como é o caso da questão “What was the original name of the Zune?”, já que o Zune, um tocador de mídia lançado em 2006, nunca teve outro nome, ambas contidas na Figura 29, resultaram em respostas ruins, já que não é possível responder essas perguntas. Já no caso de questões irrelevantes, marcadas como ruim, é possível obter uma resposta ótima no caso da resposta ser suficiente para responder esta pergunta realizada.

Como é possível observar na tabela 2, as arquiteturas atingiram bons resultados, possuindo, na média, uma maior concentração de resultados bons e ótimos e, comparando as arquiteturas, a que teve resultados melhores foi a Multi-Modelo com Gerador de Resposta.

A arquitetura Modelo Único teve resultados satisfatórios, mas, devido a necessidade do modelo ser treinado em mais de uma tarefa, resultou em um pior desempenho ao

Figura 28 – Exemplares de Questões de Acordo com sua Classificação



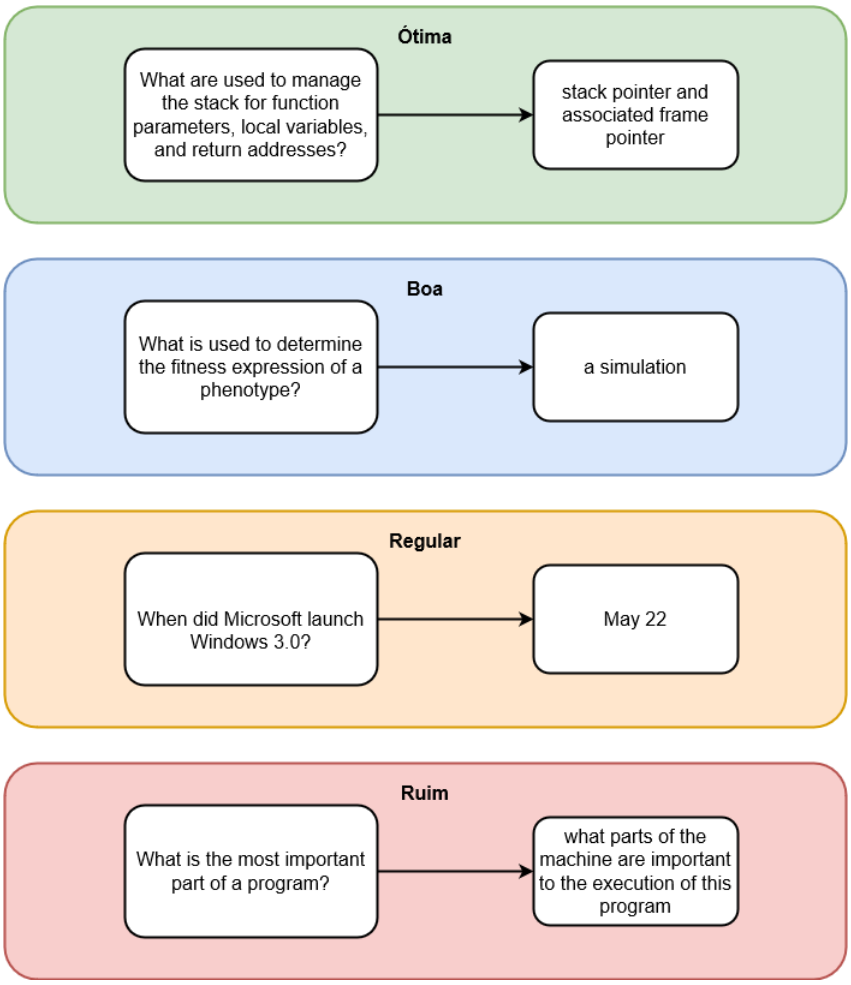
(Gerada pelo autor)

comparar com a utilização de dois modelos treinados em uma tarefa cada. Este tipo de arquitetura pode ser melhor explorada quando se há limitação no espaço em disco no ambiente aplicado, visto que, por ser apenas um modelo, ocupa menos espaço em disco que as outras arquiteturas (apenas 892 MB aproximadamente 57% menos ao comparar com a arquitetura Multi-Modelo com Gerador de Resposta).

A utilização de um modelo Gerador de Resposta na arquitetura Modelo Único, resultado na arquitetura Modelo Único com Gerador de Resposta, obteve-se os resultados esperados, ocorrendo uma melhora na qualidade das respostas presentes na saída da arquitetura, havendo um aumento na quantidade de Respostas Ótimas e Boas e diminuição das respostas Regulares e Ruins, demonstrando que tais modelos possuem uma boa capacidade de extrair as respostas presentes no texto quando apresentados com uma questão formulada pelos outros modelos.

Por fim as Arquiteturas Multi-Modelo e Multi-Modelo com Gerador de Resposta, como já apresentado, tiveram o melhor resultado na geração de questões Ótimas e Boas, e, de

Figura 29 – Exemplares de Resposta de Acordo com sua Classificação



(Gerada pelo autor)

forma semelhante a arquitetura Modelo Único, sendo adicionado o modelo Gerador de Resposta, tendo um aumento na qualidade das respostas finais.

Portanto, como demonstrado, a Arquitetura Multi-Modelo com Gerador de Resposta foi a que obteve o melhor resultado, possuindo a desvantagem de ocupar mais espaço em disco, cerca de dois gigabytes, e ter um tempo de processamento ligeiramente maior, por executar o processamento em várias redes diferentes.

Analisando os resultados obtidos por essa arquitetura separadamente por texto utilizado, é observável uma diferença clara na qualidade das questões geradas entre os textos extraídos da Wikipédia e o texto retirado do livro de sistemas operacionais, como exposto na tabela 3.

Apesar do texto de sistemas operacionais ser maior e, por isso, possuir mais questões geradas, ao analisar a qualidade das questões, houve uma concentração de resultados bons regulares e ruins, com apenas poucos exemplares de questões ótimas. Já nos demais textos, essa concentração se deu nos resultados ótimos e bons, com poucos exemplares de

Tabela 3 – Avaliação das Questões e Respostas Geradas pela Arquitetura Multi-Modelo com Gerador de Resposta por Texto

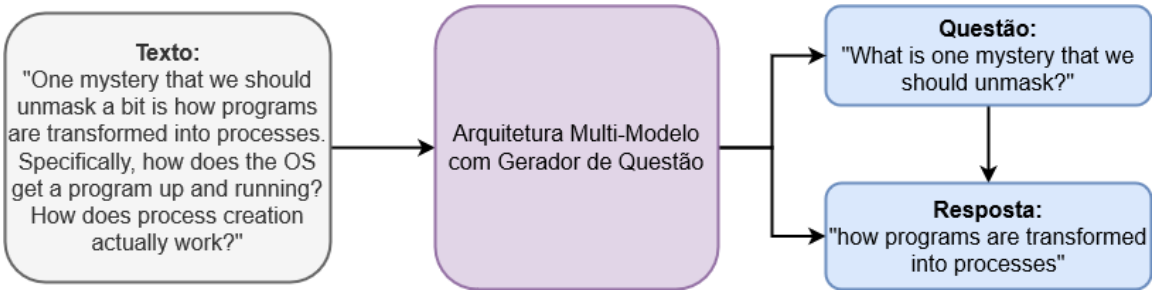
Texto	Questões				Respostas			
	Ótima	Boa	Regular	Ruim	Ótima	Boa	Regular	Ruim
Operating Systems - Process	3	13	8	9	13	2	0	5
Artigo Microsoft	8	8	3	1	16	1	1	2
Artigo Algoritmos Genéticos	7	6	2	1	13	2	0	1

questões regulares e ruins.

Percentualmente os resultados bons e ótimos totalizaram 80% no Artigo sobre a empresa Microsoft e 81,3% no artigo sobre Algoritmos Genéticos. Já no texto sobre Sistemas Operacionais, apenas 48,5% dos resultados tiveram esta classificação.

Analisando os textos para identificar a motivação desta diferença de resultados, chegou-se à conclusão de que a forma como o texto é escrito impacta diretamente nos resultados obtidos pela rede. Os artigos retirados da Wikipédia possuem uma linguagem direta, que tenta listar os fatos de forma neutra e objetiva, resultando em uma exposição mais clara das informações que são apresentadas, o que facilita o processamento a ser realizado pelos modelos geradores de questão. já no caso do capítulo retirado do livro de Sistemas Operacionais, há a utilização de uma linguagem mais amigável e acessível, o que torna o livro mais atrativo para leitores iniciantes ou que não possuem muito conhecimento técnico, mas leva a utilização de certos parágrafos irrelevantes para a geração de questão. Um exemplo deste tipo de comportamento pode ser observado na Figura 30, no qual uma questão bem construída foi gerada, com uma resposta satisfatória à pergunta realizada, mas sua relevância é baixa e por isso não é interessante para ser considerada uma boa questão.

Figura 30 – Exemplo de Questão Gerada a partir do Texto de Sistemas Operacionais



(Gerada pelo autor)

Isso demonstra que, apesar da grande capacidade de processamento da arquitetura e da qualidade dos resultados obtidos, a escolha do texto a ser utilizado impacta diretamente nestes resultados, o que torna necessário certa cautela na seleção dos textos a serem

utilizados para a geração de questões.

Apesar dos resultados mais satisfatórios nos artigos extraídos da Wikipédia, ainda existe nestes a presença de questões regulares e ruins, percentualmente sendo 15% e 5%, respectivamente, para o texto sobre a empresa Microsoft, e 12,5% e 6,3%, também respectivamente, para o texto sobre Algoritmos Genéticos, mostrando que a arquitetura desenvolvida não é perfeita e pode receber melhorias futuras focando reduzir este número.

Por fim, é preciso ressaltar a correlação entre a qualidade das questões e a qualidade das respostas geradas. Na tabela 4 identifica-se que a maior parte dos resultados ficou concentrado entre questões ótimas com respostas ótimas e questões boas com respostas ótimas. Isto demonstra que a utilização de um modelo para se gerar as respostas foi uma decisão assertiva, que permitiu obter grupos de questões e respostas relevantes.

Tabela 4 – Correlação da Qualidade das Questões e Respostas Geradas pela Arquitetura Multi-Modelo com Gerador de Resposta

Qualidade das Questões	Qualidade das Respostas			
	Ótima	Boa	Regular	Ruim
Ótima	14	2	1	1
Boa	24	2	0	1
Regular	9	0	0	4
Ruim	9	0	0	2

Apesar disto, ainda existe houve presença de respostas ruins e regulares para as questões boas e ótimas geradas. analisando cada caso chega-se à conclusão que o próprio texto utilizado deixa a desejar em relação a resposta da questão, sendo uma resposta ampla que não seria possível extrair de apenas um trecho do texto, como é o caso da utilização do parágrafo para se responder e gerar questões.

Os piores resultados de respostas ficaram concentrados nas questões regular e ruim, o que já era de se esperar visto que nessas encontram-se as questões mal formuladas ou com pouca relevância em relação ao texto e, por tanto, são questões que podem possuir respostas abertas, impossíveis de se responder com qualidade satisfatória.

Portanto, não foi possível remover totalmente a necessidade de interação na etapa de geração de atividades, mas, com a qualidade dos resultados obtidos, é possível reduzir o trabalho necessário no momento da formulação das atividades, reduzindo-o para uma etapa de verificação e validação das questões e respostas geradas em que uma validação humana auxiliará na seleção dos melhores resultados para apresentá-los aos alunos.

5.6 Utilização de Outros Sistemas

Ao longo do desenvolvimento deste trabalho, diversas divulgações sobre inteligencia artificial foram realizadas, mostrando um novo passo na capacidade dos sistemas computacionais especializados em compreender e gerar textos em linguagem natural. O lançamento de ferramentas como o chatGPT ou o Bard, ambos baseados em transformer, chamaram a atenção de diversas áreas do conhecimento por sua capacidade de se comunicar e realizar tarefas solicitadas sem a necessidade de um treinamento especifico. Isto só é possível com a grande quantidade de informação e poder computacional utilizados no treinamento deste tipo de rede.

O grande atrativo nessas ferramentas, além de sua capacidade, esta na interface fácil e amigável, no qual por meio de um chat é possível interagir com os modelos, que são hospedados pelas empresas que os desenvolveram, sendo disponibilizado com todo o treinamento já realizado. Isto permite que ate os usuários mais leigos, com nenhum conhecimento em treinamento e arquitetura de redes neurais, possam extrair resultados interessantes com estas interações. dentre estas ferramentas, a que mais se destacou foi o ChatGPT, que possui esse nome por ser baseado na arquitetura GPT (*Generative Pre-trained Transformer*) desenvolvida pela OpenAI. A arquitetura GPT tem sido amplamente reconhecida por sua capacidade de gerar texto de alta qualidade e coerência, tornando-se uma referência no campo de modelos de linguagem baseados em redes neurais. O ChatGPT traz consigo as características positivas da arquitetura GPT e as aplica em um contexto de conversação. Isso significa que o modelo é capaz de interpretar perguntas, comandos e interações em um formato de diálogo, gerando respostas relevantes e contextualmente coerentes. No entanto, é importante ressaltar que, embora o ChatGPT seja uma ferramenta poderosa, ele também possui suas limitações. O modelo não tem conhecimento específico além do que foi treinado, o que significa que suas respostas podem não ser sempre precisas ou atualizadas em relação a eventos recentes. Além disso, o modelo pode gerar respostas que parecem corretas, mas que na verdade são incorretas ou imprecisas.

Na tarefa de geração de questões o modelo se mostra altamente capaz a primeira vista, compreendendo sem dificuldades a tarefa proposta e gerando resultados coerentes e compreensíveis de acordo com o texto fornecido. Ao solicitar que o mesmo criasse 10 pares de questões e respostas baseadas no texto fornecido, este prontamente respondeu com o as questões enumeradas com suas respectivas respostas. As questões geradas se mostraram bem escritas e formuladas, mas as respostas geradas continham muitas divagações, o que indicava a utilização de outras fontes de conhecimento para responder as mesmas. A principal vantagem destes sistemas pode se tornar o principal empecilho neste cenário, pois, por possuírem uma grande base de conhecimento derivado do treinamento que inclui muitos textos, artigos e livros, este comumente acaba utilizando conhecimento que não esta presente no texto fornecido para a formulação das questões. Isto faz com que as

questões geradas nem sempre sejam apresentadas de forma a serem respondidas apenas com a utilização dos textos fornecidos, tornando o resultado pouco proveitoso para o contexto em que este trabalho está inserido.

Apesar disto, reforçando a necessidade de que a resposta da questão deveria estar contida no texto através de novas mensagens enviadas no chat, o ChatGPT foi capaz de refazer as questões com respostas extraídas diretamente deste. Esta reformulação só foi possível com a identificação feita por humanos nos resultados gerados, fazendo com que, no caso de utilização de sistemas automatizados, este problema possa passar despercebido visto que a avaliação automatizada das respostas e questões é um dos desafios principais encontrados durante o desenvolvimento deste trabalho.

5.7 Considerações Finais

Neste capítulo foi apresentado o ambiente utilizado e os resultados obtidos ao desenvolver e aplicar as arquiteturas propostas no capítulo anterior. Os testes realizados permitiram identificar os erros e acertos contidos nas arquiteturas, tendo resultante a arquitetura Multi-Modelo com Gerador de Resposta como a que melhor desempenhou nos aspectos avaliados.

6 Conclusões e Trabalhos Futuros

Neste capítulo são apresentados a forma como o tema proposto foi abordado e um resumo dos resultados obtidos, além de considerar possíveis melhorias futuras a serem desenvolvidas.

6.1 Conclusões

O objetivo principal deste trabalho foi o desenvolvimento de uma arquitetura que fosse capaz de, estando inserido em um contexto de sistemas tutores inteligentes, gerar questões a partir de origens informacionais textuais. A metodologia proposta permite a comparação entre diversas opções chegando a conclusão que a utilização de três modelos principais, um buscador de resposta, um gerador de questão e um gerador de resposta, tiveram o melhor resultado.

Ao longo deste texto se abordou inicialmente a forma como o ensino é realizado hoje e a necessidade de se atualizar este modelo de forma a considerar as diversas peculiaridades presentes nas pessoas, sendo uma abordagem inicial a divisão dos mesmos em grupos baseados na forma de ensino mais eficaz para estes indivíduos.

Desta contextualização foi possível observar a necessidade de realizar esta mudança de paradigma através da digitalização dos processos de ensino, principalmente através dos Sistemas Tutores Inteligentes que permitem aos alunos uma maior autonomia e um tratamento individualizado.

Tais sistemas no entanto necessitam de algoritmos especializados em realizar certas ações, como o caso abordado neste trabalho de geração de tarefas para serem realizadas pelos alunos.

Do estudo realizado é possível extrair algumas conclusões, apresentadas a seguir:

1. Diversas técnicas podem ser utilizadas para suprir esta necessidade, e foram classificadas em três paradigmas principais: Técnicas Semânticas e Sintática, Técnicas Semânticas e Sintáticas Apoiados por I.A. e Técnicas de *Deep Learning*.

2. Ao abordar Técnicas de *Deep Learning*, as redes do tipo *transformers* demonstraram os resultados mais promissores.

3. A falta de um dataset especifico para esta tarefa pode impactar fortemente nos resultados obtidos e nas pontuações dos benchmarks disponíveis.

4. Das quatro arquiteturas desenvolvidas para o contexto apresentado, a arquitetura Multi-Modelo com Gerador de Resposta foi definida como a mais aplicável .

Durante o texto foram apresentadas diversas técnicas presentes no paradigma de Geração de Questões que, ao serem observadas, foi possível dividi-las em três grupos principais sendo estes as Técnicas Semânticas e Sintáticas, em que a classificação semântica e sintática das informações é a principal tarefa para se gerar as questões de forma determinística de acordo com estas classificações, as Técnicas Semânticas e Sintáticas Apoiados por I.A., em que, além destas demarcações, há a presença de algum modelo de aprendizado de máquina visando reduzir algum trabalho nos algoritmos determinísticos, e as Técnicas de *Deep Learning* em que o processamento é concentrado em modelos de redes neurais que são responsáveis por gerar a questão, destacando-se as do tipo Transformer.

Foi então explorado a forma como redes do tipo Transformer são construídas e como seus mecanismos funcionam, como a forma que a arquitetura *encoder-decoder* é utilizada, os cálculos de atenção realizados e como esta abordagem resultou em novos estados da arte em diversas tarefas diferentes, além de reconhecer alguns avanços realizados no modelo ao longo dos anos.

Analisando o desempenho deste tipo de rede na tarefa de geração de questões foi constatado que a falta de um dataset construído especificamente para a tarefa é um fator que impacta tanto o treinamento das redes quanto seu resultado dos benchmarks relacionados a estes datasets, visto que, na maioria dos casos, ao realizar uma comparação direta entre a questão resultante do modelo e a questão presente no dataset podem divergir, mas, apesar disto, a questão gerada pode estar correta e ser relevante no contexto, já que no caso de datasets como o SQuAD, o intuito é a utilização destes para o treinamento de modelos especializados em responder questões e, por isso, não foi construído com o intuito de proporcionar um treinamento eficiente para modelos que busquem gerar as questões em si. A falta de um dataset específico para esta tarefa pode ser um ponto relevante a se explorar e possibilitar melhores resultados obtidos a partir do treinamento dos modelos utilizando-se deste.

Por fim foi instituído e desenvolvido quatro arquiteturas diferentes, diferenciando entre elas a quantidade de modelos de inteligência artificial presente e seu papel na arquitetura, sendo então feitas comparações ao realizar a geração de questões em três textos com tópicos e utilidades diferentes, avaliando a qualidade das questões geradas e das respostas esperadas.

A partir destes testes realizados é possível concluir que o trabalho cumpre com seu propósito ao apresentar uma arquitetura que possui resultado satisfatório, sendo nomeada de Arquitetura Multi-Modelo com Gerador de Resposta, mas é necessário reconhecer que outros fatores, como a forma que o texto é escrito ou a estrutura dos parágrafos utilizados, impactam diretamente no resultado final da arquitetura.

6.2 Trabalhos Futuros

Analisando os resultados obtidos é possivel reconhecer pontos que podem ser melhorados, sendo listados a seguir:

1. Pré-processamento do texto.
2. Filtrar questões geradas.
3. Utilização de outros sistemas para a geração de questão.

Relacionado ao início de operação das arquiteturas apresentadas está o pré-processamento do texto que será utilizado para a geração de questões. De forma muito simples, atualmente o texto é separado por parágrafo para que seja gerada uma questão para cada um destes parágrafos, no entanto formas mais sofisticadas de pré-processamento podem ser exploradas e aplicadas.

Abordagens que possam envolver outros modelos de inteligência artificial treinados para esta tarefa também podem ser exploradas neste sentido, em que seu objetivo seria a extração de frases chave dentro do texto que serão, então, convertidas em questões pelo restante da arquitetura. Estas melhorias visão melhorar a qualidade do *input* inicial, visto que este pode impactar de forma significativa a qualidade das questões geradas.

Outro ponto a ser explorado futuramente é a filtragem das questões geradas, permitindo que a arquitetura descarte questões mal formuladas ou selecione apenas as questões mais relevantes. Isto implica que formas automatizadas de qualificar estas questões sejam exploradas, apesar de se mostrarem um dos principais desafios relatados nos demais trabalhos na área. Com isto seria possível eliminar qualquer necessidade de interação humana no processo, permitindo que as falhas da arquitetura sejam sanadas através desta etapa final, garantindo, então, a qualidade das questões disponibilizadas.

Desta forma, são salientados pontos a se melhorar no início e no fim do processo de geração de questão, que, de certa forma, não impactam diretamente na arquitetura gerada, podendo ser adicionados facilmente como passos adicionais na arquitetura que foi demonstrada como a mais eficiente para este cenário.

No entanto, a utilização de outros sistemas para a realização do processamento principal da geração de questões também não deve ser descartada, visto que, apesar do pouco tempo decorrido durante este trabalho, diversas novas redes surgiram, prometendo apresentar poder de processamento e capacidades muito superiores aos seus antecessores. Assim os modelos internos contidos na arquitetura também podem ser futuramente substituídos por modelos mais recentes e com resultados melhores, o que tornaria a arquitetura mais robusta e eficiente.

6.3 Considerações Finais

A tarefa de geração de questão é uma tarefa complexa, pois envolve diversos fatores atrelados a este. superficialmente pode-se acreditar que se trata apenas de uma inversão de sentido na frase, transformando uma afirmação em uma questão, mas, apesar disto, gerar questões com sentido e que sejam relevantes é um grande desafio presente no cenário atual. Apesar de ainda não possuir um grande destaque como é o caso do paradigma de responder questões, gradualmente este vem se destacando devido ao fato que a capacidade de se gerar questões está diretamente ligada a comunicação humana e, por isso, para que se haja cada vez mais a diminuição da barreira de comunicação presente entre os sistemas digitais e as pessoas.

O ponto principal deste paradigma tem sido explorado em cenários que envolvam o aprendizado, no qual a geração de questões já é utilizada mais claramente através dos questionários feitos por professores para testar e aprofundar o conhecimento dos alunos. Neste sentido este trabalho buscou facilitar esta tarefa que recai sobre os professores, permitindo a estes se dedicarem aos demais pontos presentes na realização do ensino.

Além disso, aliado a um Sistema Tutor Inteligente, esta arquitetura pode permitir esta facilitação de forma mais intuitiva e menos complicada, sendo utilizada pelo STI como um módulo presente em sua composição.

REFERÊNCIAS

1 PIAGET, J. *The psychology of intelligence*. [S.l.]: Routledge, 2003.

2 DEWEY, J. *Democracy and education: An introduction to the philosophy of education*. [S.l.]: macmillan, 1923.

3 KOLB, D. A. *Experiential learning: Experience as the source of learning and development*. [S.l.]: FT press, 2014.

4 KOLB, D. A. *Learning style inventory technical manual*. [S.l.]: McBer Boston, 1976.

5 CARBONELL, J. R. AI in CAI: An artificial-intelligence approach to computer-assisted instruction. *IEEE transactions on man-machine systems*, IEEE, v. 11, n. 4, p. 190–202, 1970.

6 PERONAGLIO, F. F. et al. Adaptação automática de conteúdo aplicada em ambiente interativo de aprendizagem individualizada. *Revista Brasileira de Informática na Educação*, v. 31, n. 1, p. 255–270, maio 2023. Disponível em: ⟨https://sol.sbc.org.br/journals/index.php/rbie/article/view/2906⟩.

7 VASWANI, A. et al. Attention is all you need. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2017. p. 5998–6008.

8 BAHRINI, A. et al. *ChatGPT: Applications, Opportunities, and Threats*. 2023.

9 ANIL, R. et al. *PaLM 2 Technical Report*. 2023.

10 RUS, V.; CAI, Z.; GRAESSER, A. Question generation: Example of a multi-year evaluation campaign. *Proc WS on the QGSTEC*, 2008.

11 MADASU, A.; OLIVA, J.; BERTASIUS, G. Learning to retrieve videos by asking questions. In: *Proceedings of the 30th ACM International Conference on Multimedia*. New York, NY, USA: Association for Computing Machinery, 2022. (MM '22), p. 356–365. ISBN 9781450392037. Disponível em: ⟨https://doi.org/10.1145/3503161.3548361⟩.

12 LEE, C.-H. et al. Automatic question generation from children's stories for companion chatbot. In: IEEE. *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. [S.l.], 2018. p. 491–494.

13 LEITE, B. et al. Factual question generation for the portuguese language. In: IEEE. *2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*. [S.l.], 2020. p. 1–7.

14 CHALI, Y.; HASAN, S. A. Towards topic-to-question generation. *Computational Linguistics*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , v. 41, n. 1, p. 1–20, 2015.

15 PILLAI, L. R.; VEENA, G.; GUPTA, D. A combined approach using semantic role labelling and word sense disambiguation for question generation and answer extraction. In: IEEE. *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC)*. [S.l.], 2018. p. 1–6.

16 KUMAR, A. et al. Automation of question-answer generation. In: IEEE. *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT)*. [S.l.], 2021. p. 175–180.

17 OLNEY, A. M.; GRAESSER, A. C.; PERSON, N. K. Question generation from concept maps. *Dialogue & Discourse*, v. 3, n. 2, p. 75–99, 2012.

18 AGARWAL, M.; SHAH, R.; MANNEM, P. Automatic question generation using discourse cues. In: *Proceedings of the Sixth Workshop on Innovative Use of NLP for Building Educational Applications*. [S.l.: s.n.], 2011. p. 1–9.

19 RAYNAUD, T.; SUBERCAZE, J.; LAFOREST, F. Thematic question generation over knowledge bases. In: IEEE. *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. [S.l.], 2018. p. 1–8.

20 ILYA, B.; VADIM, M.; SILNOV, D. Automatic generation of questions based on semantic text analysis. In: IEEE. *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*. [S.l.], 2020. p. 1–4.

21 SHAH, R.; SHAH, D.; KURUP, L. Automatic question generation for intelligent tutoring systems. In: IEEE. *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*. [S.l.], 2017. p. 127–132.

22 AGARWAL, M.; MANNEM, P. Automatic gap-fill question generation from text books. In: *Proceedings of the sixth workshop on innovative use of NLP for building educational applications*. [S.l.: s.n.], 2011. p. 56–64.

23 ZHANG, J.; TAKUMA, J. A kanji learning system based on automatic question sentence generation. In: IEEE. *2015 international conference on asian language processing (IALP)*. [S.l.], 2015. p. 144–147.

24 KWANKAJORNKIET, C.; SUCHATO, A.; PUNYABUKKANA, P. Automatic multiple-choice question generation from thai text. In: IEEE. *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. [S.l.], 2016. p. 1–6.

25 PABITHA, P. et al. Automatic question generation system. In: IEEE. *2014 International Conference on Recent Trends in Information Technology*. [S.l.], 2014. p. 1–5.

26 KILLAWALA, A.; KHOKHLOV, I.; REZNIK, L. Computational intelligence framework for automatic quiz question generation. In: IEEE. *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2018. p. 1–8.

27 BECKER, L.; BASU, S.; VANDERWENDE, L. Mind the gap: learning to choose gaps for question generation. In: *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. [S.l.: s.n.], 2012. p. 742–751.

28 BEDNARIK, L.; KOVACS, L. Implementation and assessment of the automatic question generation module. In: IEEE. *2012 IEEE 3rd international conference on cognitive infocommunications (CogInfoCom)*. [S.l.], 2012. p. 687–690.

29 BLŠTÁK, M.; ROZINAJOVÁ, V. Building an agent for factual question generation task. In: IEEE. *2018 World symposium on digital intelligence for systems and machines (DISA)*. [S.l.], 2018. p. 143–150.

30 KOSTADINOV, S. *Understanding encoder-decoder sequence to sequence model*. Towards Data Science, 2019. Disponível em: ⟨https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346⟩.

31 KIM, Y. et al. Improving neural question generation using answer separation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2019. v. 33, n. 01, p. 6602–6609.

32 LU, X. Learning to generate questions with adaptive copying neural networks. In: *Proceedings of the 2019 International Conference on Management of Data*. [S.l.: s.n.], 2019. p. 1838–1840.

33 ZHAO, Y. et al. Paragraph-level neural question generation with maxout pointer and gated self-attention networks. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. [S.l.: s.n.], 2018. p. 3901–3910.

34 WU, K. et al. Separate answer decoding for multi-class question generation. In: IEEE. *2019 International Conference on Asian Language Processing (IALP)*. [S.l.], 2019. p. 325–330.

35 SONG, L. et al. Leveraging context information for natural question generation. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. [S.l.: s.n.], 2018. p. 569–574.

36 LIU, B. Neural question generation based on Seq2Seq. In: *Proceedings of the 2020 5th International Conference on Mathematics and Artificial Intelligence*. [S.l.: s.n.], 2020. p. 119–123.

37 NEMA, P. et al. Let's ask again: Refine network for automatic question generation. *arXiv preprint arXiv:1909.05355*, 2019.

38 WANG, Z. et al. Qg-net: a data-driven question generation model for educational content. In: *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*. [S.l.: s.n.], 2018. p. 1–10.

39 WILLIS, A. et al. Key phrase extraction for generating educational question-answer pairs. In: *Proceedings of the Sixth (2019) ACM Conference on Learning@ Scale*. [S.l.: s.n.], 2019. p. 1–10.

40 SUN, X. et al. Answer-focused and position-aware neural question generation. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. [S.l.: s.n.], 2018. p. 3930–3939.

41 YUAN, W.; HE, T.; DAI, X. Improving neural question generation using deep linguistic representation. In: *Proceedings of the Web Conference 2021*. [S.l.: s.n.], 2021. p. 3489–3500.

42 LI, X. et al. Rad: Reinforced attention decoder model on question generation. In: IEEE. *2020 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2020. p. 1–8.

43 NIO, L.; MURAKAMI, K. Intelligence is asking the right question: A study on japanese question generation. In: IEEE. *2018 IEEE Spoken Language Technology Workshop (SLT)*. [S.l.], 2018. p. 771–778.

44 LIANG, C. et al. Distractor generation with generative adversarial nets for automatically creating fill-in-the-blank questions. In: *Proceedings of the Knowledge Capture Conference*. [S.l.: s.n.], 2017. p. 1–4.

45 BAO, J. et al. Question generation with doubly adversarial nets. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, IEEE, v. 26, n. 11, p. 2230–2239, 2018.

46 LIU, B. et al. Learning to generate questions by learningwhat not to generate. In: *The World Wide Web Conference*. [S.l.: s.n.], 2019. p. 1106–1118.

47 CAO, Z.; TATINATI, S.; KHONG, A. W. Controllable question generation via sequence-to-sequence neural model with auxiliary information. In: IEEE. *2020 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2020. p. 1–7.

48 ZENG, H. et al. Improving paragraph-level question generation with extended answer network and uncertainty-aware beam search. *Information Sciences*, Elsevier, v. 571, p. 50–64, 2021.

49 GANGOPADHYAY, S.; RAVIKIRAN, S. Focused questions and answer generation by key content selection. In: IEEE. *2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)*. [S.l.], 2020. p. 45–53.

50 LING, Y. et al. Leveraging context for neural question generation in open-domain dialogue systems. In: *Proceedings of The Web Conference 2020*. [S.l.: s.n.], 2020. p. 2486–2492.

51 SINGH, J.; SHARMA, Y. Encoder-decoder architectures for generating questions. *Procedia computer science*, Elsevier, v. 132, p. 1041–1048, 2018.

52 DU, X.; CARDIE, C. Identifying where to focus in reading comprehension for neural question generation. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. [S.l.: s.n.], 2017. p. 2067–2073.

53 BALASUNDARAM, S. et al. Improving the cognitive levels of automatic generated questions using neuro-fuzzy approach in e-assessment. In: IEEE. *2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*. [S.l.], 2020. p. 454–458.

54 LELKES, A. D.; TRAN, V. Q.; YU, C. Quiz-style question generation for news stories. In: *Proceedings of the Web Conference 2021*. [S.l.: s.n.], 2021. p. 2501–2511.

55 SHAN, J. et al. Question generation for reading comprehension of language learning test:-a method using seq2seq approach with transformer model. In: IEEE. *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. [S.l.], 2019. p. 1–6.

56 YU, J. et al. Multi-hop reasoning question generation and its application. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, 2021.

57 PAN, Y. et al. Learning to generate diverse questions from keywords. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.], 2020. p. 8224–8228.

58 DU, X.; SHAO, J.; CARDIE, C. Learning to ask: Neural question generation for reading comprehension. *arXiv preprint arXiv:1705.00106*, 2017.

59 LIU, B. et al. Asking questions the human way: Scalable question-answer generation from text corpus. In: *Proceedings of The Web Conference 2020*. [S.l.: s.n.], 2020. p. 2032–2043.

60 TSAI, D. C. et al. Automatic question generation for repeated testing to improve student learning outcome. In: IEEE. *2021 International Conference on Advanced Learning Technologies (ICALT)*. [S.l.], 2021. p. 339–341.

61 KUMAR, A. et al. Automatic question-answer pair generation using deep learning. In: IEEE. *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)*. [S.l.], 2021. p. 794–799.

62 RAJPURKAR, P.; JIA, R.; LIANG, P. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

63 KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

64 GEHRING, J. et al. Convolutional sequence to sequence learning. In: PMLR. *International Conference on Machine Learning*. [S.l.], 2017. p. 1243–1252.

65 RADFORD, A. et al. Improving language understanding by generative pre-training. 2018.

66 DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

67 WANG, A. et al. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

68 RAJPURKAR, P. et al. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

69 RAFFEL, C. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

70 PROFESSOR, G. E. d. P. E. D. *Estilos de aprendizagem*. 2020. Disponível em: ⟨https://professor.escoladigital.pr.gov.br/estilos_aprendizagem⟩.

71 PASHLER, H. et al. Learning styles: Concepts and evidence. *Psychological Science in the Public Interest*, v. 9, n. 3, p. 105–119, 2008. PMID: 26162104. Disponível em: ⟨https://doi.org/10.1111/j.1539-6053.2009.01038.x⟩.

72 NEWTON, P. M.; MIAH, M. Evidence-based higher education – is the learning styles 'myth' important? *Frontiers in Psychology*, v. 8, 2017. ISSN 1664-1078. Disponível em: ⟨https://www.frontiersin.org/articles/10.3389/fpsyg.2017.00444⟩.

73 WOLF, T. et al. Transformers: State-of-the-art natural language processing. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, 2020. p. 38–45. Disponível em: ⟨https://www.aclweb.org/anthology/2020.emnlp-demos.6⟩.

74 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*. [S.l.: s.n.], 2002. p. 311–318.

75 BANERJEE, S.; LAVIE, A. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. [S.l.: s.n.], 2005. p. 65–72.

76 LIN, C.-Y. Rouge: A package for automatic evaluation of summaries. In: *Text summarization branches out*. [S.l.: s.n.], 2004. p. 74–81.

77 XIAO, D. et al. *ERNIE-GEN: An Enhanced Multi-Flow Pre-training and Fine-tuning Framework for Natural Language Generation*. 2020.

78 PATIL, S. *Question Generation*. [S.l.]: GitHub, 2020. ⟨https://github.com/patil-suraj/question_generation⟩.

79 CHAN, Y.-H.; FAN, Y.-C. A recurrent BERT-based model for question generation. In: *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*. Hong Kong, China: Association for Computational Linguistics, 2019. p. 154–162. Disponível em: ⟨https://aclanthology.org/D19-5821⟩.

80 SANH, V. et al. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

81 ARPACI-DUSSEAU, R. H.; ARPACI-DUSSEAU, A. C. *Operating systems: Three easy pieces*. [S.l.]: Arpaci-Dusseau Books, LLC, 2018.

82 Wikipedia contributors. *Genetic algorithm — Wikipedia, The Free Encyclopedia*. 2023. ⟨https://en.wikipedia.org/w/index.php?title=Genetic_algorithm&oldid=1147036946⟩. [Online; accessed 6-April-2023].

83 Wikipedia contributors. *Microsoft — Wikipedia, The Free Encyclopedia*. 2023. ⟨https://en.wikipedia.org/w/index.php?title=Microsoft&oldid=1148128940⟩. [Online; accessed 6-April-2023].

A *Texto 1 – Operating Systems: Three Easy Pieces*

Neste apêndice será apresentado inicialmente o texto extraído do livro "Operating Systems: Three Easy Pieces", mais especificamente seu capítulo 4. Na sequência apresenta-se o conjunto de questões geradas para este texto.

A.1 Texto original

In this chapter, we discuss one of the most fundamental abstractions that the OS provides to users: the process. The definition of a process, informally, is quite simple: it is a running program. The program itself is a lifeless thing: it just sits there on the disk, a bunch of instructions (and maybe some static data), waiting to spring into action. It is the operating system that takes these bytes and gets them running, transforming the program into something useful.

It turns out that one often wants to run more than one program at once; for example, consider your desktop or laptop where you might like to run a web browser, mail program, a game, a music player, and so forth. In fact, a typical system may be seemingly running tens or even hundreds of processes at the same time. Doing so makes the system easy to use, as one never need be concerned with whether a CPU is available; one simply runs programs. Hence our challenge:

The OS creates this illusion by virtualizing the CPU. By running one process, then stopping it and running another, and so forth, the OS can promote the illusion that many virtual CPUs exist when in fact there is only one physical CPU (or a few). This basic technique, known as time sharing of the CPU, allows users to run as many concurrent processes as they would like; the potential cost is performance, as each will run more slowly if the CPU(s) must be shared.

To implement virtualization of the CPU, and to implement it well, the OS will need both some low-level machinery and some high-level intelligence. We call the low-level machinery mechanisms; mechanisms are low-level methods or protocols that implement a needed piece of functionality. For example, we'll learn later how to implement a context switch, which gives the OS the ability to stop running one program and start running another on a given CPU; this time-sharing mechanism is employed by all modern OSes.

On top of these mechanisms resides some of the intelligence in the OS, in the form of policies. Policies are algorithms for making some kind of decision within the OS. For example, given a number of possible programs to run on a CPU, which program

should the OS run? A scheduling policy in the OS will make this decision, likely using historical information (e.g., which program has run more over the last minute?), workload knowledge (e.g., what types of programs are run), and performance metrics (e.g., is the system optimizing for interactive performance, or throughput?) to make its decision.

The abstraction provided by the OS of a running program is something we will call a process. As we said above, a process is simply a running program; at any instant in time, we can summarize a process by taking an inventory of the different pieces of the system it accesses or affects during the course of its execution.

To understand what constitutes a process, we thus have to understand its machine state: what a program can read or update when it is running. At any given time, what parts of the machine are important to the execution of this program?

One obvious component of machine state that comprises a process is its memory. Instructions lie in memory; the data that the running program reads and writes sits in memory as well. Thus the memory that the process can address (called its address space) is part of the process.

Also part of the process's machine state are registers; many instructions explicitly read or update registers and thus clearly they are important to the execution of the process.

Note that there are some particularly special registers that form part of this machine state. For example, the program counter (PC) (sometimes called the instruction pointer or IP) tells us which instruction of the program will execute next; similarly a stack pointer and associated frame pointer are used to manage the stack for function parameters, local variables, and return addresses.

Finally, programs often access persistent storage devices too. Such I/O information might include a list of the files the process currently has open.

Though we defer discussion of a real process API until a subsequent chapter, here we first give some idea of what must be included in any interface of an operating system. These APIs, in some form, are available on any modern operating system.

Create: An operating system must include some method to create new processes. When you type a command into the shell, or double-click on an application icon, the OS is invoked to create a new process to run the program you have indicated.

Destroy: As there is an interface for process creation, systems also provide an interface to destroy processes forcefully. Of course, many processes will run and just exit by themselves when complete; when they don't, however, the user may wish to kill them, and thus an interface to halt a runaway process is quite useful.Wait: Sometimes it is useful to wait for a process to stop running; thus some kind of waiting interface is often provided.

Miscellaneous Control: Other than killing or waiting for a process, there are sometimes other controls that are possible. For example, most operating systems provide some kind of method to suspend a process (stop it from running for a while) and then resume it (continue it running).

Status: There are usually interfaces to get some status information about a process as well, such as how long it has run for, or what state it is in.

One mystery that we should unmask a bit is how programs are transformed into processes. Specifically, how does the OS get a program up and running? How does process creation actually work?

The first thing that the OS must do to run a program is to load its code and any static data (e.g., initialized variables) into memory, into the address space of the process. Programs initially reside on disk (or, in some modern systems, flash-based SSDs) in some kind of executable format; thus, the process of loading a program and static data into memory requires the OS to read those bytes from disk and place them in memory somewhere (as shown in Figure 4.1).

In early (or simple) operating systems, the loading process is done eagerly, i.e., all at once before running the program; modern OSes perform the process lazily, i.e., by loading pieces of code or data only as they are needed during program execution. To truly understand how lazy loading of pieces of code and data works, you'll have to understand more about the machinery of paging and swapping, topics we'll cover in the future when we discuss the virtualization of memory. For now, just remember that before running anything, the OS clearly must do some work to get the important program bits from disk into memory.

Once the code and static data are loaded into memory, there are a few other things the OS needs to do before running the process. Some memory must be allocated for the program's run-time stack (or just stack). As you should likely already know, C programs use the stack for local variables, function parameters, and return addresses; the OS allocates this memory and gives it to the process. The OS will also likely initialize the stack with arguments; specifically, it will fill in the parameters to the main() function, i.e., argc and the argv array.

The OS may also allocate some memory for the program's heap. In C programs, the heap is used for explicitly requested dynamically-allocated data; programs request such space by calling malloc() and free it explicitly by calling free(). The heap is needed for data structures such as linked lists, hash tables, trees, and other interesting data structures. The heap will be small at first; as the program runs, and requests more memory via the malloc() library API, the OS may get involved and allocate more memory to the process to help satisfy such calls.

The OS will also do some other initialization tasks, particularly as related to input/output (I/O). For example, in UNIX systems, each process by default has three open file descriptors, for standard input, output, and error; these descriptors let programs easily read input from the terminal and print output to the screen. We'll learn more about I/O, file descriptors, and the like in the third part of the book on persistence.

By loading the code and static data into memory, by creating and initializing a stack,

and by doing other work as related to I/O setup, the OS has now (finally) set the stage for program execution. It thus has one last task: to start the program running at the entry point, namely main(). By jumping to the main() routine (through a specialized mechanism that we will discuss next chapter), the OS transfers control of the CPU to the newly-created process, and thus the program begins its execution.

Now that we have some idea of what a process is (though we will continue to refine this notion), and (roughly) how it is created, let us talk about the different states a process can be in at a given time. The notion that a process can be in one of these states arose in early computer systems . In a simplified view, a process can be in one of three states:

Running: In the running state, a process is running on a processor. This means it is executing instructions.

Ready: In the ready state, a process is ready to run but for some reason the OS has chosen not to run it at this given moment.

Blocked: In the blocked state, a process has performed some kind of operation that makes it not ready to run until some other event takes place. A common example: when a process initiates an I/O request to a disk, it becomes blocked and thus some other process can use the processor.

If we were to map these states to a graph, we would arrive at the diagram in Figure 4.2. As you can see in the diagram, a process can be moved between the ready and running states at the discretion of the OS. Being moved from ready to running means the process has been scheduled; being moved from running to ready means the process has been descheduled. Once a process has become blocked (e.g., by initiating an I/O operation), the OS will keep it as such until some event occurs (e.g., I/O completion); at that point, the process moves to the ready state again (and potentially immediately to running again, if the OS so decides). Let's look at an example of how two processes might transition through some of these states. First, imagine two processes running, each of which only use the CPU (they do no I/O). In this case, a trace of the state of each process might look like this (Figure 4.3).

In this next example, the first process issues an I/O after running for some time. At that point, the process is blocked, giving the other process a chance to run. Figure 4.4 shows a trace of this scenario.

More specifically, Process0 initiates an I/O and becomes blocked waiting for it to complete; processes become blocked, for example, when reading from a disk or waiting for a packet from a network. The OS recognizes Process0 is not using the CPU and starts running Process1. While Process1 is running, the I/O completes, moving Process0 back to ready. Finally, Process1 finishes, and Process0 runs and then is done.Note that there are many decisions the OS must make, even in this simple example. First, the system had to decide to run Process1 while Process0 issued an I/O; doing so improves resource utilization by keeping the CPU busy. Second, the system decided not to switch back to

Process0 when its I/O completed; it is not clear if this is a good decision or not. What do you think? These types of decisions are made by the OS scheduler, a topic we will discuss a few chapters in the future.The OS is a program, and like any program, it has some key data structures that track various relevant pieces of information. To track the state of each process, for example, the OS likely will keep some kind of process list for all processes that are ready and some additional information to track which process is currently running. The OS must also track, in some way, blocked processes; when an I/O event completes, the OS should make sure to wake the correct process and ready it to run again. Figure 4.5 shows what type of information an OS needs to track about each process in the xv6 kernel. Similar process structures exist in "real" operating systems such as Linux, Mac OS X, or Windows; look them up and see how much more complex they are.

From the figure, you can see a couple of important pieces of information the OS tracks about a process. The register context will hold, for a stopped process, the contents of its registers. When a process is stopped, its registers will be saved to this memory location; by restoring these registers (i.e., placing their values back into the actual physical registers), the OS can resume running the process. We'll learn more about this technique known as a context switch in future chapters.

You can also see from the figure that there are some other states a process can be in, beyond running, ready, and blocked. Sometimes a system will have an initial state that the process is in when it is being created. Also, a process could be placed in a final state where it has exited but has not yet been cleaned up (in UNIX-based systems, this is called the zombie state). This final state can be useful as it allows other processes (usually the parent that created the process) to examine the return code of the process and see if the just-finished process executed successfully (usually, programs return zero in UNIX-based systems when they have accomplished a task successfully, and non-zero otherwise). When finished, the parent will make one final call (e.g., wait()) to wait for the completion of the child, and to also indicate to the OS that it can clean up any relevant data structures that referred to the now-extinct process.

We have introduced the most basic abstraction of the OS: the process. It is quite simply viewed as a running program. With this conceptual view in mind, we will now move on to the nitty-gritty: the low-level mechanisms needed to implement processes, and the higher-level policies required to schedule them in an intelligent way. By combining mechanisms and policies, we will build up our understanding of how an operating system virtualizes the CPU.

A.2 Questões Geradas

Q1. *What is the operating system's process?*

R1. Transforming the program into something useful.

Q2. *How many processes can a typical system run at the same time?*

R2. Tens or even hundreds.

Q3. *What is the basic technique used by OS to create the illusion that many virtual CPUs exist?*

R3. Time sharing of the CPU.

Q4. *What are low-level methods or protocols that implement a needed piece of functionality called?*

R4. Low-level machinery mechanisms.

Q5. *What is the intelligence in the OS called?*

R5. Policies.

Q6. *How can we summarize a process?*

R6. By taking an inventory of the different pieces of the system.

Q7. *What is the most important part of a program?*

R7. What parts of the machine.

Q8. *What is the name of the memory that a process can address?*

R8. Address space.

Q9. *What is an important part of the machine state?*

R9. Registers.

Q10. *What are used to manage the stack for function parameters, local variables, and return addresses?*

R10. Stack pointer and associated frame pointer.

Q11. *What do programs often access?*

R11. Persistent storage devices.

Q12. *When is the real process API discussed?*

R12. A subsequent chapter.

Q13. *What are two ways an operating system can create new processes?*

R13. Type a command into the shell, or double-click on an application icon.

Q14. *What is the name of the interface that allows a system to destroy processes?*

R14. Destroy.

Q15. *What is a method to do?*

R15. Suspend a process.

Q16. *What are some examples of status information?*

R16. How long it has run for, or what state it is in.

Q17. *What is one mystery that we should unmask?*

R17. How programs are transformed into processes.

Q18. *What does loading a program and static data into memory require?*

R18. The OS to read those bytes from disk.

Q19. *What type of operating system does the loading process happen eagerly?*

R19. Early (or simple) operating systems.

Q20. *What happens when the memory is allocated to the process?*

R20. The OS allocates this memory and gives it to the process.

Q21. *What happens when a program requests more memory?*

R21. The OS may get involved.

Q22. *When will we learn more about I/O, file descriptors, and the like?*

R22. In the third part of the book on persistence.

Q23. *What is the first step in the execution of a program?*

R23. The OS transfers control of the CPU to the newly-created process.

Q24. *What is the simplified view of a process?*

R24. A process can be in one of three states.

Q25. *What is a process doing when it is running?*

R25. Executing instructions.

Q26. *What is ready to run?*

R26. A process.

Q27. *When does a process become blocked?*

R27. When a process initiates an I/O request to a disk.

Q28. *What is the diagram of the state of a process?*

R28. Figure 4.2.

Q29. *What happens in the next example?*

R29. The first process issues an I/O after running for some time.

Q30. *What process is not using the CPU?*

R30. Process0.

Q31. *What is the technique that restores registers called?*

R31. Context switch.

Q32. *What is the zombie state?*

R32. A final state where it has exited but has not yet been cleaned up.

Q33. *What is the most basic abstraction of the OS?*

R33. The process.

B Texto 2 – *Genetic algorithm*

Aqui apresentamos inicialmente o texto sobre algoritmos genéticos extraído da Wikipédia, seguido das questões geradas a partir dele.

B.1 Texto original

In computer science and operations research, a genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. Some examples of GA applications include optimizing decision trees for better performance, solving sudoku puzzles, hyperparameter optimization, etc.

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, organisms, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires: a genetic representation of the solution domain, a fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits (also called bit set or bit string). Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic program-

ming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

During each successive generation, a portion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise.

In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g. computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

The next step is to generate a second generation population of solutions from those selected, through a combination of genetic operators: crossover (also called recombination), and mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research suggests that more than two "parents" generate higher quality chromosomes.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally, the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Opinion is divided over the importance of crossover versus mutation. There are many references in Fogel (2006) that support the importance of mutation-based search. Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms.

It is worth tuning parameters such as the mutation probability, crossover probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions, unless elitist selection is employed. An adequate population size ensures sufficient genetic diversity for the problem at hand, but can lead to a waste of computational resources if set to a value larger than required.

In addition to the main operators above, other heuristics may be employed to make the calculation faster or more robust. The speciation heuristic penalizes crossover between candidate solutions that are too similar; this encourages population diversity and helps prevent premature convergence to a less optimal solution.This generational process is repeated until a termination condition has been reached. Common terminating conditions are: A solution is found that satisfies minimum criteria; Fixed number of generations reached; Allocated budget (computation time/money) reached; The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results; Manual inspection; Combinations of the above.

Genetic algorithms are simple to implement, but their behavior is difficult to understand. In particular, it is difficult to understand why these algorithms frequently succeed at generating solutions of high fitness when applied to practical problems. The building block hypothesis (BBH) consists of: A description of a heuristic that performs adaptation by identifying and recombining "building blocks", i.e. low order, low defining-length schemata with above average fitness; A hypothesis that a genetic algorithm performs adaptation by implicitly and efficiently implementing this heuristic.

B.2 Questões Geradas

Q1. *What are some biologically inspired operators?*

R1. Mutation, crossover and selection.

Q2. *What is evolved toward better solutions in a genetic algorithm?*

R2. A population of candidate solutions.

Q3. *When does the algorithm end?*

R3. When either a maximum number of generations has been produced.

Q4. *What does a typical genetic algorithm require to evaluate the solution domain?*

R4. A fitness function.

Q5. *What are two other names for an array of bits?*

R5. Bit set or bit string.

Q6. *What is generated randomly?*

R6. The initial population.

Q7. *What is selected to breed a new generation?*

R7. A portion of the existing population.

Q8. *What is the fitness of the solution?*

R8. The sum of values of all objects in the knapsack.

Q9. *What is used to determine the fitness expression of a phenotype?*

R9. A simulation.

Q10. *What is another name for crossover?*

R10. Recombination.

Q11. *What does research suggest that more than two parents produce?*

R11. Higher quality chromosomes.

Q12. *What does less fit solutions ensure genetic diversity within?*

R12. The genetic pool of the parents.

Q13. *What is the main genetic operator?*

R13. Crossover and mutation.

Q14. *What parameters are worth tuning to find reasonable settings for the problem class being worked on?*

R14. Mutation probability, crossover probability and population size.

Q15. *What can be used to make the calculation faster or more robust?*

R15. Other heuristics.

Q16. *What is difficult to understand but simple to implement?*

R16. Genetic algorithms.

C Texto 3 – Microsoft

Finalmente, neste último apêndice apresenta-se o texto sobre a Microsoft, também extraído da Wikipédia, junto com as questões geradas a partir dele.

C.1 Texto original

Microsoft Corporation is an American multinational technology corporation headquartered in Redmond, Washington. Microsoft's best-known software products are the Windows line of operating systems, the Microsoft Office suite, and the Internet Explorer and Edge web browsers. Its flagship hardware products are the Xbox video game consoles and the Microsoft Surface lineup of touchscreen personal computers. Microsoft ranked No. 14 in the 2022 Fortune 500 rankings of the largest United States corporations by total revenue;it was the world's largest software maker by revenue as of 2022. It is considered as one of the Big Five American information technology companies, alongside Alphabet (parent company of Google), Amazon, Apple, and Meta (formerly Facebook).

Microsoft was founded by Bill Gates and Paul Allen on April 4, 1975, to develop and sell BASIC interpreters for the Altair 8800. It rose to dominate the personal computer operating system market with MS-DOS in the mid-1980s, followed by Windows. The company's 1986 initial public offering (IPO), and subsequent rise in its share price, created three billionaires and an estimated 12,000 millionaires among Microsoft employees. Since the 1990s, it has increasingly diversified from the operating system market and has made a number of corporate acquisitions, their largest being the acquisition of LinkedIn for $26.2 billion in December 2016, followed by their acquisition of Skype Technologies for $8.5 billion in May 2011.

As of 2015, Microsoft is market-dominant in the IBM PC compatible operating system market and the office software suite market, although it has lost the majority of the overall operating system market to Android. The company also produces a wide range of other consumer and enterprise software for desktops, laptops, tabs, gadgets, and servers, including Internet search (with Bing), the digital services market (through MSN), mixed reality (HoloLens), cloud computing (Azure), and software development (Visual Studio).

Steve Ballmer replaced Gates as CEO in 2000, and later envisioned a 'devices and services' strategy. This unfolded with Microsoft acquiring Danger Inc. in 2008, entering the personal computer production market for the first time in June 2012 with the launch of the Microsoft Surface line of tablet computers, and later forming Microsoft Mobile through the acquisition of Nokia's devices and services division. Since Satya Nadella took over as CEO in 2014, the company has scaled back on hardware and has instead focused

on cloud computing, a move that helped the company's shares reach its highest value since December 1999.

Earlier dethroned by Apple in 2010, in 2018 Microsoft reclaimed its position as the most valuable publicly traded company in the world.[10] In April 2019, Microsoft reached the trillion-dollar market cap, becoming the third U.S. public company to be valued at over $1 trillion after Apple and Amazon respectively. As of 2022, Microsoft has the fourth-highest global brand valuation.

Microsoft has been criticized for its monopolistic practices and the company's software has been criticized for problems with ease of use, robustness, and security.

Childhood friends Bill Gates and Paul Allen sought to make a business using their skills in computer programming. In 1972, they founded Traf-O-Data, which sold a rudimentary computer to track and analyze automobile traffic data. Gates enrolled at Harvard University while Allen pursued a degree in computer science at Washington State University, though he later dropped out to work at Honeywell. The January 1975 issue of Popular Electronics featured Micro Instrumentation and Telemetry Systems's (MITS) Altair 8800 microcomputer, which inspired Allen to suggest that they could program a BASIC interpreter for the device. Gates called MITS and claimed that he had a working interpreter, and MITS requested a demonstration. Allen worked on a simulator for the Altair while Gates developed the interpreter, and it worked flawlessly when they demonstrated it to MITS in March 1975 in Albuquerque, New Mexico. MITS agreed to distribute it, marketing it as Altair BASIC. Gates and Allen established Microsoft on April 4, 1975, with Gates as CEO, and Allen suggested the name 'Micro-Soft', short for micro-computer software. In August 1977, the company formed an agreement with ASCII Magazine in Japan, resulting in its first international office of ASCII Microsoft. Microsoft moved its headquarters to Bellevue, Washington, in January 1979.

Microsoft entered the operating system (OS) business in 1980 with its own version of Unix called Xenix, but it was MS-DOS that solidified the company's dominance. IBM awarded a contract to Microsoft in November 1980 to provide a version of the CP/M OS to be used in the IBM Personal Computer (IBM PC). For this deal, Microsoft purchased a CP/M clone called 86-DOS from Seattle Computer Products which it branded as MS-DOS, although IBM rebranded it to IBM PC DOS. Microsoft retained ownership of MS-DOS following the release of the IBM PC in August 1981. IBM had copyrighted the IBM PC BIOS, so other companies had to reverse engineer it in order for non-IBM hardware to run as IBM PC compatibles, but no such restriction applied to the operating systems. Microsoft eventually became the leading PC operating systems vendor. The company expanded into new markets with the release of the Microsoft Mouse in 1983, as well as with a publishing division named Microsoft Press. Paul Allen resigned from Microsoft in 1983 after developing Hodgkin's lymphoma. Allen claimed in Idea Man: A Memoir by the Co-founder of Microsoft that Gates wanted to dilute his share in the company when he

was diagnosed with Hodgkin's disease because he did not think that he was working hard enough. Allen later invested in low-tech sectors, sports teams, commercial real estate, neuroscience, private space flight, and more.

Microsoft released Windows on November 20, 1985, as a graphical extension for MS-DOS, espite having begun jointly developing OS/2 with IBM the previous August. Microsoft moved its headquarters from Bellevue to Redmond, Washington, on February 26, 1986, and went public on March 13, with the resulting rise in stock making an estimated four billionaires and 12,000 millionaires from Microsoft employees. Microsoft released its version of OS/2 to original equipment manufacturers (OEMs) on April 2, 1987. In 1990, the Federal Trade Commission examined Microsoft for possible collusion due to the partnership with IBM, marking the beginning of more than a decade of legal clashes with the government. Meanwhile, the company was at work on Microsoft Windows NT, which was heavily based on their copy of the OS/2 code. It shipped on July 21, 1993, with a new modular kernel and the 32-bit Win32 application programming interface (API), making it easier to port from 16-bit (MS-DOS-based) Windows. Microsoft informed IBM of Windows NT, and the OS/2 partnership deteriorated.

In 1990, Microsoft introduced the Microsoft Office suite which bundled separate applications such as Microsoft Word and Microsoft Excel. On May 22, Microsoft launched Windows 3.0, featuring streamlined user interface graphics and improved protected mode capability for the Intel 386 processor, and both Office and Windows became dominant in their respective areas.

On July 27, 1994, the Department of Justice's Antitrust Division filed a competitive impact statement that said: "Beginning in 1988 and continuing until July 15, 1994, Microsoft induced many OEMs to execute anti-competitive 'per processor licenses. Under a per-processor license, an OEM pays Microsoft a royalty for each computer it sells containing a particular microprocessor, whether the OEM sells the computer with a Microsoft operating system or a non-Microsoft operating system. In effect, the royalty payment to Microsoft when no Microsoft product is being used acts as a penalty, or tax, on the OEM's use of a competing PC operating system. Since 1988, Microsoft's use of per processor licenses has increased."

Following Bill Gates' internal "Internet Tidal Wave memo" on May 26, 1995, Microsoft began to redefine its offerings and expand its product line into computer networking and the World Wide Web. With a few exceptions of new companies, like Netscape, Microsoft was the only major and established company that acted fast enough to be a part of the World Wide Web practically from the start. Other companies like Borland, WordPerfect, Novell, IBM and Lotus, being much slower to adapt to the new situation, would give Microsoft a market dominance. The company released Windows 95 on August 24, 1995, featuring pre-emptive multitasking, a completely new user interface with a novel start button, and 32-bit compatibility; similar to NT, it provided the Win32 API. Windows 95

came bundled with the online service MSN, which was at first intended to be a competitor to the Internet, and (for OEMs) Internet Explorer, a Web browser. Internet Explorer has not bundled with the retail Windows 95 boxes, because the boxes were printed before the team finished the Web browser, and instead were included in the Windows 95 Plus! pack. Backed by a high-profile marketing campaign and what The New York Times called "the splashiest, most frenzied, most expensive introduction of a computer product in the industry's history", Windows 95 quickly became a success. Branching out into new markets in 1996, Microsoft and General Electric's NBC unit created a new 24/7 cable news channel, MSNBC. Microsoft created Windows CE 1.0, a new OS designed for devices with low memory and other constraints, such as personal digital assistants. In October 1997, the Justice Department filed a motion in the Federal District Court, stating that Microsoft violated an agreement signed in 1994 and asked the court to stop the bundling of Internet Explorer with Windows.

On January 13, 2000, Bill Gates handed over the CEO position to Steve Ballmer, an old college friend of Gates and employee of the company since 1980, while creating a new position for himself as Chief Software Architect. Various companies including Microsoft formed the Trusted Computing Platform Alliance in October 1999 to (among other things) increase security and protect intellectual property through identifying changes in hardware and software. Critics decried the alliance as a way to enforce indiscriminate restrictions over how consumers use software, and over how computers behave, and as a form of digital rights management: for example, the scenario where a computer is not only secured for its owner but also secured against its owner as well. On April 3, 2000, a judgment was handed down in the case of United States v. Microsoft Corp., calling the company an "abusive monopoly". Microsoft later settled with the U.S. Department of Justice in 2004. On October 25, 2001, Microsoft released Windows XP, unifying the mainstream and NT lines of OS under the NT codebase. The company released the Xbox later that year, entering the video game console market dominated by Sony and Nintendo. In March 2004 the European Union brought antitrust legal action against the company, citing it abused its dominance with the Windows OS, resulting in a judgment of €497 million ($613 million) and requiring Microsoft to produce new versions of Windows XP without Windows Media Player: Windows XP Home Edition N and Windows XP Professional N. In November 2005, the company's second video game console, the Xbox 360, was released. There were two versions, a basic version for $299.99 and a deluxe version for $399.99.

Increasingly present in the hardware business following Xbox, Microsoft 2006 released the Zune series of digital media players, a successor of its previous software platform Portable Media Center. These expanded on previous hardware commitments from Microsoft following its original Microsoft Mouse in 1983; as of 2007 the company sold the best-selling wired keyboard (Natural Ergonomic Keyboard 4000), mouse (IntelliMouse), and desktop webcam (LifeCam) in the United States. That year the company also

launched the Surface "digital table", later renamed PixelSense.

Released in January 2007, the next version of Windows, Vista, focused on features, security and a redesigned user interface dubbed Aero. Microsoft Office 2007, released at the same time, featured a "Ribbon" user interface which was a significant departure from its predecessors. Relatively strong sales of both products helped to produce a record profit in 2007. The European Union imposed another fine of €899 million ($1.4 billion) for Microsoft's lack of compliance with the March 2004 judgment on February 27, 2008, saying that the company charged rivals unreasonable prices for key information about its workgroup and backoffice servers.[59] Microsoft stated that it was in compliance and that "these fines are about the past issues that have been resolved". 2007 also saw the creation of a multi-core unit at Microsoft, following the steps of server companies such as Sun and IBM.

Gates retired from his role as Chief Software Architect on June 27, 2008, a decision announced in June 2006, while retaining other positions related to the company in addition to being an advisor for the company on key projects. Azure Services Platform, the company's entry into the cloud computing market for Windows, launched on October 27, 2008. On February 12, 2009, Microsoft announced its intent to open a chain of Microsoft-branded retail stores, and on October 22, 2009, the first retail Microsoft Store opened in Scottsdale, Arizona; the same day Windows 7 was officially released to the public. Windows 7's focus was on refining Vista with ease-of-use features and performance enhancements, rather than an extensive reworking of Windows.

As the smartphone industry boomed in the late 2000s, Microsoft had struggled to keep up with its rivals in providing a modern smartphone operating system, falling behind Apple and Google-sponsored Android in the United States. As a result, in 2010 Microsoft revamped their aging flagship mobile operating system, Windows Mobile, replacing it with the new Windows Phone OS that was released in October that year. It used a new user interface design language, codenamed "Metro", which prominently used simple shapes, typography, and iconography, utilizing the concept of minimalism. Microsoft implemented a new strategy for the software industry, providing a consistent user experience across all smartphones using the Windows Phone OS. It launched an alliance with Nokia in 2011 and Microsoft worked closely with the company to co-develop Windows Phone, but remained partners with long-time Windows Mobile OEM HTC. Microsoft is a founding member of the Open Networking Foundation started on March 23, 2011. Fellow founders were Google, HP Networking, Yahoo!, Verizon Communications, Deutsche Telekom and 17 other companies. This nonprofit organization is focused on providing support for a cloud computing initiative called Software-Defined Networking. The initiative is meant to speed innovation through simple software changes in telecommunications networks, wireless networks, data centers, and other networking areas.When Microsoft went public and launched its initial public offering (IPO) in 1986, the opening stock price was $21;

after the trading day, the price closed at $27.75. As of July 2010, with the company's nine stock splits, any IPO shares would be multiplied by 288; if one were to buy the IPO today, given the splits and other factors, it would cost about 9 cents. The stock price peaked in 1999 at around 119(60.928, adjusting for splits). The company began to offer a dividend on January 16, 2003, starting at eight cents per share for the fiscal year followed by a dividend of sixteen cents per share the subsequent year, switching from yearly to quarterly dividends in 2005 with eight cents a share per quarter and a special one-time payout of three dollars per share for the second quarter of the fiscal year. Though the company had subsequent increases in dividend payouts, the price of Microsoft's stock remained steady for years.

Standard & Poor's and Moody's Investors Service have both given a AAA rating to Microsoft, whose assets were valued at $41 billion as compared to only $8.5 billion in unsecured debt. Consequently, in February 2011 Microsoft released a corporate bond amounting to $2.25 billion with relatively low borrowing rates compared to government bonds. For the first time in 20 years Apple Inc. surpassed Microsoft in Q1 2011 quarterly profits and revenues due to a slowdown in PC sales and continuing huge losses in Microsoft's Online Services Division (which contains its search engine Bing). Microsoft profits were $5.2 billion, while Apple Inc. profits were $6 billion, on revenues of $14.5 billion and $24.7 billion respectively.Microsoft's Online Services Division has been continuously loss-making since 2006 and in Q1 2011 it lost $726 million. This follows a loss of $2.5 billion for the year 2010.

On July 20, 2012, Microsoft posted its first quarterly loss ever, despite earning record revenues for the quarter and fiscal year, with a net loss of $492 million due to a writedown related to the advertising company aQuantive, which had been acquired for $6.2 billion back in 2007. As of January 2014, Microsoft's market capitalization stood at $314B, making it the 8th largest company in the world by market capitalization. On November 14, 2014, Microsoft overtook ExxonMobil to become the second most-valuable company by market capitalization, behind only Apple Inc. Its total market value was over 410B - with the stock price hitting $50.04 a share, the highest since early 2000. In 2015, Reuters reported that Microsoft Corp had earnings abroad of $76.4 billion which were untaxed by the Internal Revenue Service. Under U.S. law, corporations don't pay income tax on overseas profits until the profits are brought into the United States.

In November 2018, the company won a $480 million military contract with the U.S. government to bring augmented reality (AR) headset technology into the weapon repertoires of American soldiers. The two-year contract may result in follow-on orders of more than 100,000 headsets, according to documentation describing the bidding process. One of the contract's tag lines for the augmented reality technology seems to be its ability to enable '25 bloodless battles before the 1st battle', suggesting that actual combat training is going to be an essential aspect of the augmented reality headset capabilities

C.2 Questões Geradas

Q1. *Where is Microsoft Corporation headquartered?*

R1. Redmond, Washington

Q2. *When was Microsoft founded?*

R2. April 4, 1975

Q3. *What operating system has Microsoft lost the majority of the overall operating system market to?*

R3. Android

Q4. *Who replaced Gates as CEO in 2000?*

R4. Steve Ballmer

Q5. *What company dethroned Microsoft in 2010?*

R5. Apple

Q6. *What problems has Microsoft had with its software?*

R6. Ease of use, robustness, and security

Q7. *Who were two of the two friends who started Traf-O-Data?*

R7. Bill Gates and Paul Allen

Q8. *What was the name of Microsoft's own version of Unix?*

R8. Xenix

Q9. *When was Windows released?*

R9. November 20, 1985

Q10. *When did Microsoft launch Windows 3.0?*

R10. May 22

Q11. *When did the Department of Justice's Antitrust Division file a competitive impact statement?*

R11. July 27, 1994

Q12. *What happened to Microsoft following Bill Gates' memo?*

R12. Microsoft violated an agreement signed in 1994

Q13. *Who was the CEO of Microsoft in 2000?*

R13. Steve Ballmer

Q14. *What was the original name of the Zune?*

R14. Microsoft Mouse

Q15. *When was the next version of Windows released?*

R15. January 2007

Q16. *When did Gates retire from his position as Chief Software Architect?*

R16. June 27, 2008

Q17. *When did the smartphone industry boom?*

R17. Late 2000s

Q18. *What was Microsoft's unsecured debt worth?*

R18. $8.5 billion

Q19. *When did Microsoft post its first quarterly loss ever?*

R19. July 20, 2012

Q20. *What is an essential aspect of AR?*

R20. Actual combat training