UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Tecnologia

Andrey Toshiro Okamura

Detecção de Erros em Sentenças Textuais para Análise STPA com Classificação via BERT

Limeira
2025

Andrey Toshiro Okamura

Detecção de Erros em Sentenças Textuais para Análise STPA com Classificação via BERT

Dissertação apresentada à Faculdade de Tecnologia da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Tecnologia, na área de Sistemas de Informação e Comunicação.

Orientadora: Profa. Dra. Ana Estela Antunes da Silva

Este trabalho corresponde à versão final da Dissertação defendida por Andrey Toshiro Okamura e orientada pela Profa. Dra. Ana Estela Antunes da Silva.

Limeira
2025

Ficha catalográfica
Universidade Estadual de Campinas (UNICAMP)
Biblioteca da Faculdade de Tecnologia
Felipe de Souza Bueno - CRB 8/8577

Okamura, Andrey Toshiro, 2001-
Ok1d Detecção de erros em sentenças textuais para análise STPA com classificação via BERT / Andrey Toshiro Okamura. – Limeira, SP : [s.n.], 2025.

Orientador: Ana Estela Antunes da Silva.
Dissertação (mestrado) – Universidade Estadual de Campinas (UNICAMP), Faculdade de Tecnologia.

1. Aprendizado de máquina. 2. Classificação. 3. Segurança. I. Silva, Ana Estela Antunes da, 1965-. II. Universidade Estadual de Campinas (UNICAMP). Faculdade de Tecnologia. III. Título.

Informações complementares

Título em outro idioma: Detection of errors in textual sentences for STPA analysis with BERT classification
Palavras-chave em inglês:
Machine learning
Classification
Safety
Área de concentração: Sistemas de Informação e Comunicação
Titulação: Mestre em Tecnologia
Banca examinadora:
Ana Estela Antunes da Silva [Orientador]
Guilherme Palermo Coelho
Carlos Henrique Netto Lahoz
Data de defesa: 03-02-2025
Programa de Pós-Graduação: Tecnologia

Objetivos de Desenvolvimento Sustentável (ODS)
Não se aplica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0009-0006-9724-6745
- Currículo Lattes do autor: http://lattes.cnpq.br/0732591610095347

FOLHA DE APROVAÇÃO

Profa. Dra. Ana Estela Antunes da Silva
Presidente da Comissão Julgadora

Prof. Dr. Guilherme Palermo Coelho
Faculdade de Tecnologia - FT/UNICAMP

Prof. Dr. Carlos Henrique Netto Lahoz
Instituto Tecnológico de Aeronáutica - ITA

Ata da defesa, assinada pelos membros da Comissão Examinadora, encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria de Pós-graduação da Faculdade de Tecnologia.

Agradecimentos

Esta dissertação é o fruto do trabalho de muitos anos, e se tornou possível graças ao suporte de muitas pessoas.

Inicialmente, gostaria de agradecer à minha família. Mesmo em terras distantes, apoiaram a minha decisão de iniciar o mestrado e me suportaram incondicionalmente durante esta jornada.

Agradeço aos amigos Danilo Soares e Natanael Nunes, grandes companheiros que jogam videogames comigo para refrescar a mente em momentos de descontração.

Agradeço ao parceiro Gabriel Nogueira Pacheco, que participou do mesmo projeto de pesquisa. Foi uma grande fonte de inspiração em diversos momentos, e as trocas de ideias permitiram alcançar os melhores resultados em ambos os trabalhos.

Agradeço aos Professores Guilherme Palermo Coelho e Carlos Henrique Netto Lahoz, que participaram da banca de qualificação e forneceram sugestões muito úteis para aprimorar a metodologia e os resultados do trabalho.

Por fim, agradeço à orientadora Profa. Dra. Ana Estela Antunes da Silva e ao Coorientador Prof. Dr. Luiz Eduardo Galvão Martins, que me guiou a mim e ao meu trabalho em todos os momentos para que o trabalho se torne digno de uma pesquisa de mestrado.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Resumo

A *System-Theoretic Process Analysis* (STPA) é uma técnica de análise de perigos destinada a sistemas complexos, que investiga a interação dos componentes do sistema para encontrar perigos. Por ser uma técnica de muitas etapas, a STPA exige um esforço manual considerável, o que motivou esta pesquisa. O objetivo desta pesquisa é automatizar a validação de textos gerados durante o primeiro passo da análise STPA, chamado de "Definir o propósito da análise", em que se definem as Perdas (objetos de valor aos *stakeholders* em que perdas são inaceitáveis), os Perigos (condições do sistema que, em situações de pior caso, podem levar às perdas) e as Restrições (condições do sistema que devem ser satisfeitas para evitar perigos) do sistema. Estes registros textuais são sentenças de uma única linha que representam cada aspecto crucial do sistema, e devem ser levados em consideração para identificação de perigos e definição de regras. Este trabalho propõe um pipeline de modelos de aprendizado de máquina chamado BEDS: *BERT Error Detection for STPA*, (ou Detecção de erros com BERT para STPA), que tem o propósito de detectar erros e sugerir correções de sentenças para texto gerado durante o primeiro passo da análise STPA. Este pipeline é composto por 4 passos baseados no modelo de linguagem *Bidirectional Encoder Representations from Transformers* (BERT), onde a entrada é uma lista de sentenças e rótulos (com as classes de Perdas, Perigos e Restrições), e a saída do último passo são duas listas com as sentenças classificadas entre corretas ou incorretas com base no guia oficial de STPA (*STPA Handbook*). A lista de sentenças consideradas incorretas são acompanhadas pelo tipo de erro de escrita detectado, a probabilidade dos erros, e uma lista de sugestões de sentenças corretas para correção, para cada sentença incorreta. Os modelos foram avaliados com métricas de classificação de aprendizado de máquina: acurácia, precisão, sensibilidade e F1-Score. O primeiro passo de classificação, que determina a classe da sentença, atingiu 95,20% de acurácia. O segundo passo, que determina se a sentença é correta ou incorreta, atingiu a média entre os classificadores de 88,51% de acurácia. O terceiro passo, que detecta o tipo de erro presente na sentença incorreta atingiu a média de 83,44% de acurácia. O quarto e último passo utiliza um modelo de similaridade de sentenças para sugerir sentenças corretas com base em um *dataset* verificado por especialistas de STPA. As contribuições deste trabalho incluem a criação do pipeline de detecção de erros BEDS e a criação de um *dataset* de sentenças da análise STPA para treinamento e teste dos modelos de aprendizado de máquina.

palavras-chaves: *Aprendizado de Máquina, Classificação, Segurança, BERT, LLM, STPA*

Abstract

The System-Theoretic Process Analysis (STPA) is a hazard analysis technique for complex systems that investigates the interactions between components to find hazards. STPA requires a lot of manual effort because it is a multi-step technique, which motivated this research. This work aims to automate the text validation generated during the first step of the STPA analysis, called "Define the Purpose of the analysis" where the Losses (something of value which a loss is unacceptable to stakeholders), Hazards (a set of conditions that together with worst-case environmental conditions will lead to a Loss) and Constraints (system conditions that need to be satisfied to prevent hazards) are defined. These textual records are single-line sentences that represent the system's crucial aspects and should be taken into account when identifying hazards or defining rules. This work proposes a new machine learning model Pipeline called BEDS: BERT Error Detection for STPA, which aims to detect errors and suggest corrections to textual sentences generated during the first step of the STPA analysis. This Pipeline is composed of four steps that use the Bidirectional Encoder Representations from Transformers (BERT) language model, in which the input is a list of sentences and labels (with the Loss, Hazard, and Constraint classes), and the outputs are two lists that contain the sentences classified as either correct or incorrect based on the STPA Handbook. The list of incorrect sentences also contains the type of sentence error detected, the probability of each type of error, and a list of correct sentence suggestions for each incorrect sentence. The models were evaluated with machine learning classification metrics: accuracy, precision, recall, and F1-Score. The first classification step, which determines the class of a sentence, achieved an accuracy of 95.20%. The second step, which determines whether a sentence is correct or incorrect, achieved an accuracy average of 88.51% between the classifiers of each class. The third step, which detects the type of error in the incorrect sentences, achieved an accuracy average of 83.44%. The fourth and last step uses a sentence similarity model to suggest correct sentences based on a dataset validated by STPA specialists. The contributions of this work include the creation of the BEDS error detection Pipeline, and the creation of a STPA analysis sentence dataset for training and testing machine learning models.

keywords: *Machine learning, Classification, Safety, BERT, LLM, STPA*

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AEB	Automated Emergency Braking
BEDS	BERT Error Detection for STPA
BERT	Bidirectional Encoder Representations from Transformers
C2V	Context2Vec
CoHA	Co-Hazard Analysis
ConOps	Concept of Operations
CoVe	Context-Vectors
ELMo	Embeddings from Language Models
EOS	End Of Sentence
FMEA	Análise de Modos de Falha e Efeitos
FTA	Árvore de Análise de Falhas
GLUE	General Language Understanding Evaluation
LLM	Large Language Models
LN	Linguagem Natural
LSTM	Long Short Term Memory
MLM	Masked Language Model
NSP	Next Sentence Prediction
PLN	Processamento de Linguagem Natural
Q&A	Perguntas e Respostas
RNN	Rede Neural Recorrente
SOS	Start Of Sentence
STAMP	System-Theoretic Accident Model and Process
STPA	System-Theoretic Process Analysis
TF-IDF	Term Frequency - Inverse Document Frequency

Sumário

Capítulo 1

Introdução

Sistemas críticos são sistemas que não podem apresentar falhas durante sua execução, pois sua falha pode resultar em perda de valores como vidas humanas ou equipamentos.

Nestas situações, a análise de perigos se torna essencial para identificar problemas no sistema e prevenir a ocorrência de acidentes. Muitos métodos de análise tradicionais se baseiam em cadeias de eventos e funcionam bem para identificar falhas, principalmente em componentes físicos e sistemas simples (LEVESON, N., 2004).

Modelos baseados em eventos, como a Árvore de Análise de Falhas (FTA) e Análise de Modos de Falha e Efeitos (FMEA) foram desenvolvidos há mais de 50 anos e não garantem a segurança completa dos sistemas complexos atuais (ABDULKHALEQ; WAGNER; LEVESON, N., 2015). Estes métodos são lineares devido ao relacionamento de causalidade entre os componentes e a busca do fator causador do acidente. Esta linearidade dificulta a identificação de problemas não-lineares, como interações humanas e erros de software. Também podem ser subjetivos ao decidirem o que causou o acidente, pois condições e eventos podem ser sempre adicionados antes do problema (LEVESON, N., 2004), ou seja, se há uma condição que causou um acidente, pode haver outra condição anterior que facilitou a ocorrência do acidente, e a subjetividade está relacionada com a experiência do analista em encontrar problemas.

Existem fatores que limitam a cobertura de métodos de análise de perigos tradicionais em sistemas mais recentes, como: a criação de novas tecnologias (tecnologias são criadas mais rapidamente do que as técnicas de análise conseguem acompanhar e manterem-se atualizadas); mudança da natureza e novos tipos de falhas (de falhas por motivos predominantemente físicas

para falhas digitais); aumento de complexidade do sistema; e aumento da interação entre o sistema e humanos ou fatores externos (LEVESON, N., 2004).

Leveson (LEVESON, N., 2004) propôs o *System-Theoretic Accident Model and Process (STAMP)*, que define como alvo de análise o sistema como um todo, ao invés de considerar componentes isolados. De acordo com este modelo, acidentes ocorrem devido à falta de cumprimento de regras de segurança ou controles de sistema inadequados. Portanto, ao invés de prevenir falhas pontuais de componentes, o STAMP busca impor restrições ao sistema para que ele se comporte de forma segura.

A *System-Theoretic Process Analysis (STPA)* (LEVESON, N. G., 2012), é uma técnica de análise de perigos em sistemas críticos derivada do STAMP. Assim como o STAMP, a STPA tem por foco a identificação de perigos nos relacionamentos e interações entre cada componente do sistema, e encontra problemas não percebidos por outros métodos. Este modelo expande a área de análise de acidentes ao considerar também softwares e fatores humanos como componentes do sistema (ABDULKHALEQ; WAGNER; LEVESON, N., 2015).

A STPA é composta por quatro passos (LEVESON, N.; THOMAS, 2018):

1. Definir o propósito da análise - Identificar três conceitos fundamentais do sistema, dados por: Perdas ou *Losses*, que são os objetos ou entidades de valor aos stakeholders que não devem sofrer perdas, como vidas humanas, equipamentos ou missão; Perigos ao nível de sistema ou *System-Level Hazards*, que são condições do sistema que, em situações de ambiente de pior caso, podem levar às Perdas; e Restrições ao nível de sistema ou *System-Level Constraints*, que são estados do sistema que devem ser cumpridos para evitar a ocorrência de Perigos. Este passo resulta em um documento de análise STPA que contém textos curtos representantes de requisitos de perdas, restrições e perigos do sistema. Neste trabalho, estes textos serão chamados de sentenças de análise STPA.

2. Modelar a estrutura de controle - Definir a estrutura de controle composta por pelo menos cinco elementos: os controladores, as ações de controle (ações para controlar processos), o *feedback*, demais *inputs* (além do controle e *feedback*) e o processo controlado. Por meio da definição da estrutura de controle serão detectados os relacionamentos entre os componentes do sistema.

3. Identificar ações de controle inseguras - Descrever ações de controle que, em um determinado contexto podem resultar em Perigos, tal que existem quatro formas de se

tornarem inseguras. As formas são: "Não prover uma ação de controle", "Prover uma ação que causa perigos", "Prover uma ação muito cedo, muito tarde ou em ordem equivocada", e "Ação dura por muito tempo ou é parada antes do esperado".

4. Identificar cenários de perdas - Descrever os fatores que podem levar às ações de controle inseguras ou à não execução de ações de controle, e resultam em cenários que podem gerar perdas.

O esforço necessário para verificar o texto gerado aumenta proporcionalmente ao tamanho da análise STPA realizada. Neste cenário, o uso de uma ferramenta para auxiliar o processo de verificação pode se tornar útil.

Os avanços da tecnologia de inteligência artificial nos últimos 6 anos trouxeram uma ferramenta chamada de *Large Language Models (LLM)*, utilizada para processar conteúdos de texto relacionados a tarefas de Processamento de Linguagem Natural (PLN). Esta evolução se iniciou com a introdução dos modelos *transformers* (VASWANI et al., 2017) na comunidade científica. Este modelo permitiu a máquina compreender Linguagem Natural (LN) e o contexto em textos com desempenho no estado da arte. Posteriormente, derivações dos modelos *transformers* foram feitas acessíveis ao público pelos modelos baseados em *decoders* como o *Generative Pre-Trained Transformer* (GPT) da OpenAI.

Outro LLM conhecido é o *Bidirectional Encoder Representations from Transformers* (BERT) (DEVLIN et al., 2018), um modelo baseado em *encoders* mais adequado para tarefas como Perguntas e Respostas e classificação de texto, que atingiu resultados no estado da arte em sua publicação.

O uso de LLM na análise STPA é um conceito relativamente recente, e os trabalhos relacionados se referem principalmente à geração de análise STPA com a assistência de modelos geradores de texto como o Chat-GPT (DIEMERT; WEBER, 2023; QI et al., 2023; CHARALAMPIDOU; ZELESKIDIS; DOKAS, 2024).

Esta pesquisa tem foco no primeiro passo da análise STPA, "Definir o propósito da Análise". Durante este passo são definidas frases, ou sentenças que representam aspectos críticos do sistema, como as Perdas, os Perigos e as Restrições do sistema. Quando estas sentenças são geradas por alguma ferramenta auxiliar, ou caso o analista deseje verificar as sentenças geradas durante a análise, existe a possibilidade de que as sentenças não estejam conformes, ou seja, estejam em formato inadequado ou com conteúdo não esperado por uma análise STPA.

1.1 Resultados preliminares

Durante a defesa de qualificação, foi apresentado um experimento preliminar de classificação de sentenças do primeiro passo análise STPA. A intenção deste experimento foi comparar algoritmos de classificação tradicionais com algoritmos de classificação baseado em *transformers*, mais especificamente o distil-BERT.

Como a arquitetura *transformer* supostamente possui maior capacidade de compreender o contexto da sentença e a Linguagem Natural, se espera que o mesmo tenha desempenho melhor em comparação com classificadores tradicionais.

Os dados foram divididos aleatoriamente em conjuntos de treino (80%) e teste (20%), e a acurácia e F1-Score dos classificadores foram registrados na Tabela 1.1.

Tabela 1.1: Acurácia e F1-score de classificadores.

Algoritmo	Acurácia	F1-Score
Support Vector Machines	84.70%	84.19%
Naïve Bayes	78.82%	78.15%
Decision Tree	81.56%	81.10%
distil-BERT	**93.72%**	**93.72%**

O distil-BERT, baseado em *transformer*, apresentou os melhores resultados com 93.72% de acurácia e F1-score, e se sobressaiu em relação aos classificadores tradicionais. A acurácia e F1-score estão em faixas de porcentagem próximas em todos os algoritmos, que indica a capacidade de classificação dos modelos independentemente da distribuição de classes do *dataset* preliminar.

1.2 Objetivos

O objetivo desta pesquisa é automatizar o primeiro passo da análise STPA, a fim de reduzir o esforço necessário para a geração do documento de análise STPA.

Os objetivos específicos desta pesquisa são:

- Criação da base de dados de sentenças geradas durante o primeiro passo da análise STPA;
- Treinamento e teste de um modelo de arquitetura *transformer* para verificação de sentenças da análise STPA em suas respectivas classes;

- Treinamento e teste de modelos de arquitetura *transformer* para classificação de sentenças entre corretas e incorretas, de acordo com as recomendações do guia de STPA;
- Treinamento e teste de modelos de arquitetura *transformer* para detecção de tipos de erro de escrita presentes em sentenças incorretas;
- Treinamento e teste de um modelo de arquitetura *sentence-transformer* para sugerir sentenças corretas por meio de cálculo de similaridade de sentenças.

Para isso, este trabalho propõe um pipeline de classificação de sentenças chamado *BERT Error Detection for STPA (BEDS)* (detecção de erros para análise STPA utilizando BERT). Este pipeline é composto por diversos modelos baseados em arquitetura *transformer* para auxiliar na detecção de erros de escrita em sentenças textuais geradas durante o primeiro passo da análise STPA. Esta arquitetura se mostrou eficaz para diversas tarefas de classificação de texto (FIELDS; CHOVANEC; MADIRAJU, 2024), que serão utilizadas pelo pipeline.

1.3 Justificativa

Embora o *STPA Handbook* tenha sido publicado em 2018, e suas citações venham aumentando desde então, ainda existem poucos trabalhos acadêmicos sobre a incorporação de aprendizado de máquina para atividades de STPA.

Além disso, ferramentas existentes atualmente têm como objetivo principal auxiliar a execução da análise STPA (como preencher campos e planilhas com os detalhes do sistema) (CARNIEL; BEZERRA; HIRATA, 2023). Portanto, ainda não há uma aplicação que utilize aprendizado de máquina para avaliar as sentenças geradas durante o processo de análise STPA.

Esta pesquisa de mestrado foi realizada em parceria com a Universidade Federal de São Paulo (UNIFESP), por meio de outra dissertação de mestrado, e com a Embraer.

1.4 Estrutura do documento

Este documento está estruturado da seguinte forma: o Capítulo 2 contém uma descrição da revisão de literatura e contextualização dos estudos relacionados; o Capítulo 3 descreve a

metodologia proposta para criação de uma base de dados e a criação do pipeline de classificação de sentenças BEDS; o Capítulo 4 descreve os experimentos realizados; o Capítulo 5 apresenta a conclusão e os trabalhos futuros deste trabalho.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão abordados os seguintes tópicos: A técnica *System-Theoretic Process Analysis* que será utilizada na pesquisa e Processamento de Linguagem Natural e Aprendizado de Máquina, para apresentar as tecnologias utilizadas. Por fim, os trabalhos relacionados à aplicação de aprendizado de máquina em análise STPA.

2.1 System-Theoretic Process Analysis

System-Theoretic Process Analysis (STPA) é uma técnica de análise de perigos voltada para sistemas complexos, muitas vezes compostos por uma combinação de componentes físicos e software. Estes sistemas são concebidos através de um processo de engenharia de sistemas, onde há passos como a concepção do produto, engenharia de requisitos, desenvolvimento do sistema, testes e, por fim, a integração do sistema. A STPA é capaz de realizar uma análise de alto nível desde as fases iniciais do processo, e com sucessivos detalhamentos nas fases seguintes. Isto é possível, pois a STPA considera as necessidades e objetivos de *stakeholders* como ponto de partida, e não apenas os componentes funcionais.

A análise STPA é feita a partir de documentos textuais gerados durante a engenharia de sistemas, em muitos casos a partir do documento de *Concept of Operations (ConOps)*, gerado na fase inicial da concepção do produto.

Segundo McGregor (2005) (citado por FLEMING; LEVESON, N., 2015), um ConOps "geralmente inclui uma declaração das metas e objetivos do sistema; estratégias, táticas, políticas e restrições afetando o sistema; organizações, atividades e interações entre participantes e operadores; e um processo operacional para implantar o sistema". Neste

documento estão descritas as necessidades dos *stakeholders*, e parte das informações podem ser extraídas para realizar uma análise STPA.

O documento final gerado pela STPA contém quatro passos: 1. Definir o propósito da análise, 2. Modelar a estrutura de controle, 3. Identificar ações de controle inseguras e 4. Identificar cenários de perdas (LEVESON, N.; THOMAS, 2018). No passo 1, o qual é o foco desta pesquisa, se busca identificar aspectos fundamentais do sistema, como objetos de interesse aos *stakeholders*, situações de risco envolvendo esses objetos e o limite do sistema.

Inicialmente é feita a identificação de Perdas (ou *losses*). Perdas são quaisquer objetos de valor (*stakes*), sejam eles físicos ou abstratos, em que perdas ou danos são inaceitáveis aos *stakeholders*. Podem ser incluídos ferimentos ou perda de vida humana, danos a propriedades, perda de missão ou reputação, entre outros objetos de valor. Perdas podem ser definidas ao identificar os *stakeholders*, como clientes, usuários, operadores; levantar objetos de valor ou objetivos que buscam atingir, como vida humana ou missão; e finalmente converter estes objetos em perdas, ao combinar o *stake* com a palavra de perda (por exemplo "*Loss of* "ou "*Damage to*"). Exemplos de Perdas podem ser vistas na Figura 2.1.

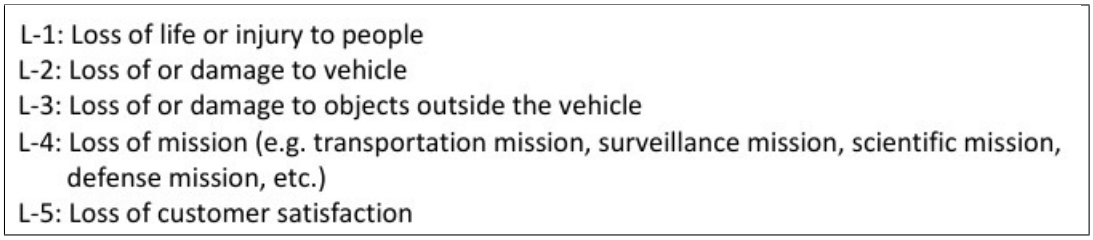
L-1: Loss of life or injury to people
L-2: Loss of or damage to vehicle
L-3: Loss of or damage to objects outside the vehicle
L-4: Loss of mission (e.g. transportation mission, surveillance mission, scientific mission, defense mission, etc.)
L-5: Loss of customer satisfaction

Figura 2.1: Exemplo de sentenças de Perda (Loss) em STPA.
Fonte: (LEVESON, N.; THOMAS, 2018)

Após as Perdas, são identificados os Perigos em nível de sistema (ou *System-Level hazards*), que são estados ou condições do sistema, que em conjunto com situações do ambiente de pior caso, podem levar às perdas. Para a STPA, o sistema é considerado um conjunto de componentes que agem em união para atingir um objetivo. O analista deve decidir qual componente deve ser incluído ou não, definindo um limite do sistema. Ao definir este limite, a diferenciação entre Perdas e Perigos é facilitada, pois as perdas envolvem aspectos do sistema em que o operador tem controle parcial ou nenhum controle, enquanto os perigos estão em partes do sistema em que se tem possibilidade de controle, e se busca evitá-los.

Os perigos devem seguir três critérios para sua definição. O primeiro critério requer que os perigos sejam estados ou condições do sistema, que estejam dentro do controle do operador (e não condições de ambiente externas), e que não sejam causas em nível de componentes. Em segundo, os perigos devem levar a perdas apenas em alguma situação de pior caso, pois podem existir estados do sistema que em condições de funcionamento normal não haja perigo. Por fim, perigos devem ser estados ou condições do sistema que devem ser prevenidos, e sejam diferentes do comportamento esperado.

É esperado, em uma sentença de perigo, a menção de <sistema> e uma <condição não segura>, sem importância da ordem de ocorrência. Para facilitar a rastreabilidade, é recomendado que a mesma sentença inclua um <identificador de perigo>, um código representando a sentença, e <link para perdas>, um conjunto de identificadores de perdas relacionadas ao perigo atual.

<Hazard specification> = <System> & <Unsafe Condition> & <Link to Losses>
E.g. H-1 = Aircraft violate minimum separation standards in flight [L-1, L-2, L-4, L-5]

Figura 2.2: Exemplo de uma sentença de Perigo (Hazard) em STPA.
Fonte: (LEVESON, N.; THOMAS, 2018)

<Hazard> = <System> & <Unsafe Condition> & <Link to Losses>
<System-level Constraint> = <System> & <Condition to Enforce> & <Link to Hazards>

H-1: Aircraft violate minimum separation standards [L-1, L-2, L-4, L-5]
SC-1: Aircraft must satisfy minimum separation standards from other aircraft and objects [H-1]

<System-level Constraint> = If <hazard> occurs, then <what needs to be done to prevent or minimize a loss> & <Link to Hazards>

SC-3: If aircraft violate minimum separation, then the violation must be detected and measures taken to prevent collision [H-1]

Figura 2.3: Exemplos de sentenças de Restrição (Constraint) em STPA.
Fonte: (LEVESON, N.; THOMAS, 2018)

Estes termos podem ser combinados como a Figura 2.2. Nesta figura, o <identificador de perigo> é dado por "H-1", o <sistema> da análise é dado por "*Aircraft*", a <condição não segura> é dada por "*violate minimum separation standards in flight*", e os <links para perdas> são dados por uma lista de identificadores "[L-1, L-2, L-4, L-5]".

As Restrições em nível de sistema (ou *System-Level Constraints*) especificam comportamentos ou condições do sistema que devem ser satisfeitos para prevenir Perigos (e consequentemente as Perdas). A definição de restrições pode ser considerada uma inversão

das sentenças definidas para Perigos, realizando a conversão do trecho de <condição não segura> para uma <condição (segura) a ser cumprida>, como ilustrado no exemplo superior da Figura 2.3. Além disso, restrições podem ser instruções de como minimizar perdas caso um perigo ocorra, ilustrado no exemplo inferior da Figura 2.3. Neste caso, é levantado um <perigo>, dado por "*aircraft violate minimum separation*", e <o que deve ser feito para minimizar a perda>, dado por "*the violation must be detected and measures taken to prevent collision*".

2.2 Processamento de Linguagem Natural e Aprendizado de Máquina

A área de Processamento de Linguagem Natural (PLN) passou por grandes avanços nos últimos anos. O surgimento da arquitetura de modelos tipo *transformer* permitiu o avanço em diversas tarefas de PLN, ao solucionar problemas existentes em modelos de redes neurais que eram consideradas estado da arte até o momento.

Nas seções seguintes serão abordados os temas de PLN, a arquitetura *transformers*, o modelo *Bidirectional Encoder Representations from Transformers (BERT)*, a classificação de texto por aprendizado de máquina, que é o foco desta pesquisa, e os *Large Language Models (LLMs)*, modelos geradores de texto utilizados nos experimentos.

2.2.1 Processamento de Linguagem Natural

A Linguagem Natural (LN) é o nome dado à linguagem utilizada para a comunicação entre pessoas, seja por texto ou fala. Hoje em dia utilizamos diversas ferramentas úteis que envolvem PLN, como tradução para outra língua, conversão de fala para texto, sumarização e paráfrase de texto. A LN possui um formato de palavras sequenciais, capazes de transmitir ideias em uma sentença organizada. Nesta sentença estão agregadas regras gramaticais e contexto, que são facilmente compreendidos por humanos em uma conversa. Computadores, no entanto, são capazes apenas de processar valores numéricos, e não capturam informações que surgem devido ao contexto e posicionamento de palavras.

As primeiras tentativas de uso de máquina para processar LN foram para tradução de línguas, em que se usava um dicionário como intermediário para traduzir palavra por palavra. No entanto, essas tentativas resultaram em falhas, pois a gramática não era levada

em consideração durante a tradução (JOHRI et al., 2021). O surgimento do aprendizado de máquina acelerou a evolução das técnicas de PLN, devido aos algoritmos mais complexos e à maior capacidade computacional disponível e exigida.

Para que a máquina consiga interpretar LN é necessário converter os dados textuais em uma representação numérica, chamada de *embedding*. Existem diversas formas de representar uma sentença, divididas em três categorias principais: representação à base de regras, representações estatísticas e representações utilizando redes neurais (PATIL et al., 2023).

As tentativas iniciais de representação foram baseadas em regras gramaticais ou padrões. Neste método, padrões, expressões lógicas e combinações de palavras eram codificadas extensivamente para encontrar a sentença desejada. Este tipo de representação é limitada pela flexibilidade das regras e não considera o contexto da sentença, o que a torna inadequada para PLN. Estas representações eram criadas por meio de expressões regulares (regex), ou expressões *if-else* aninhadas para definir regras lógicas de sentenças (PATIL et al., 2023).

Posteriormente, as técnicas estatísticas em conjunto com algoritmos de classificação tradicionais se tornaram mais frequentes. Técnicas como *One-Hot Encoding* (JAMELL IVOR SAMUELS, 2024) e *Bag of Words* (HARRIS, 1954) utilizam uma representação de matriz esparsa, em que os atributos são todas as palavras presentes no vocabulário, e as tuplas representam a presença ou a quantidade de aparições da palavra em uma sentença. No entanto, estas técnicas apresentam limitações, como a dimensão da representação se tornar muito grande de acordo com o vocabulário, ou a esparsidade da representação (a presença de muitos zeros em uma matriz) se tornar uma desvantagem em alguns casos. Além disso, estas representações são apenas um mapeamento das palavras em uma matriz, sem considerar o contexto ou a relação com as demais palavras da sentença.

Técnicas baseadas em frequência, como o *Term Frequency - Inverse Document Frequency (TF-IDF)* (PATIL et al., 2023) calculam a relevância de uma palavra em relação às demais. Esta técnica calcula o produto de duas partes: TF, ou a frequência de um termo em uma sentença, e IDF, ou a quantidade de sentenças em que o termo aparece. Nesta técnica, o valor aumenta de acordo com a frequência de uma palavra na sentença, mas é penalizado se for um termo muito comum entre todas as sentenças. Isto resulta em uma matriz de valores contínuos e formada para incluir informação sobre a importância de um termo, além da presença ou não

da palavra, resolvendo problemas das técnicas anteriores. Mesmo assim, algoritmos de PLN ainda não eram capazes de obter informação sobre o contexto e a relação entre as palavras.

O uso de Redes neurais resultou em um grande avanço para o entendimento do contexto de LN por uma máquina. Algoritmos de *embedding* como *Context2Vec (C2V)* (MELAMUD; GOLDBERGER; DAGAN, 2016), *Context-Vectors (CoVe)* (MCCANN et al., 2017) e *Embeddings from Language Models (ELMo)* (PETERS et al., 2018) utilizam adaptações do modelo *Long Short Term Memory (LSTM)*, um tipo de Rede Neural Recorrente (RNN), para gerar representações contextualizadas, com informação residual de palavras mais distantes de uma sentença. Isto é possível devido ao uso de *gates*, ou "portas", que controlam o fluxo de informação dentro da rede neural, e decidem quais informações de pouco valor a rede pode esquecer, quais novas informações devem ser acrescentadas, e o que deve se tornar a saída da camada atual. No entanto, as RNN, e por consequência LSTM, requerem maior capacidade computacional quanto maior for o texto de entrada, e ainda existe a possibilidade da perda de contexto quanto mais distante forem duas palavras no texto.

Nos últimos anos, o surgimento de algoritmos de aprendizado profundo e modelos recentes como a arquitetura *transformer* elevaram ainda mais o estado da arte para tarefas de PLN.

2.2.2 Modelos de arquitetura *Transformer*

A arquitetura *transformer* é uma arquitetura proposta por pesquisadores da Google, no artigo "*Attention Is All You Need*"(VASWANI et al., 2017), retratada na Figura 2.4. A arquitetura é composta por dois componentes principais, o *Encoder*, à esquerda, e o *Decoder*, à direita, e é utilizado principalmente em tarefas de PLN. Estes componentes serão explicados nas seções seguintes.

Até o momento de sua publicação, o estado da arte para tarefas de PLN, como tradução de línguas, eram trabalhos utilizando redes neurais como as Redes Neurais Recorrentes (RNN) junto com um mecanismo chamado de mecanismo de atenção (BAHDANAU; CHO; BENGIO, 2016).

Já o modelo *transformer* utiliza apenas o mecanismo de *Self-Attention*, ou auto-atenção, para obter o contexto de uma palavra em relação a outras dentro de uma sentença, sem a necessidade de recorrência como em redes neurais.

Isto solucionou diversos problemas sofridos por modelos RNN, como o tempo de processamento e a perda de contexto em sequências de texto maiores, e também trouxe

novidades, como a maior capacidade de captar o contexto de cada palavra, e a capacidade de cálculos em paralelo.

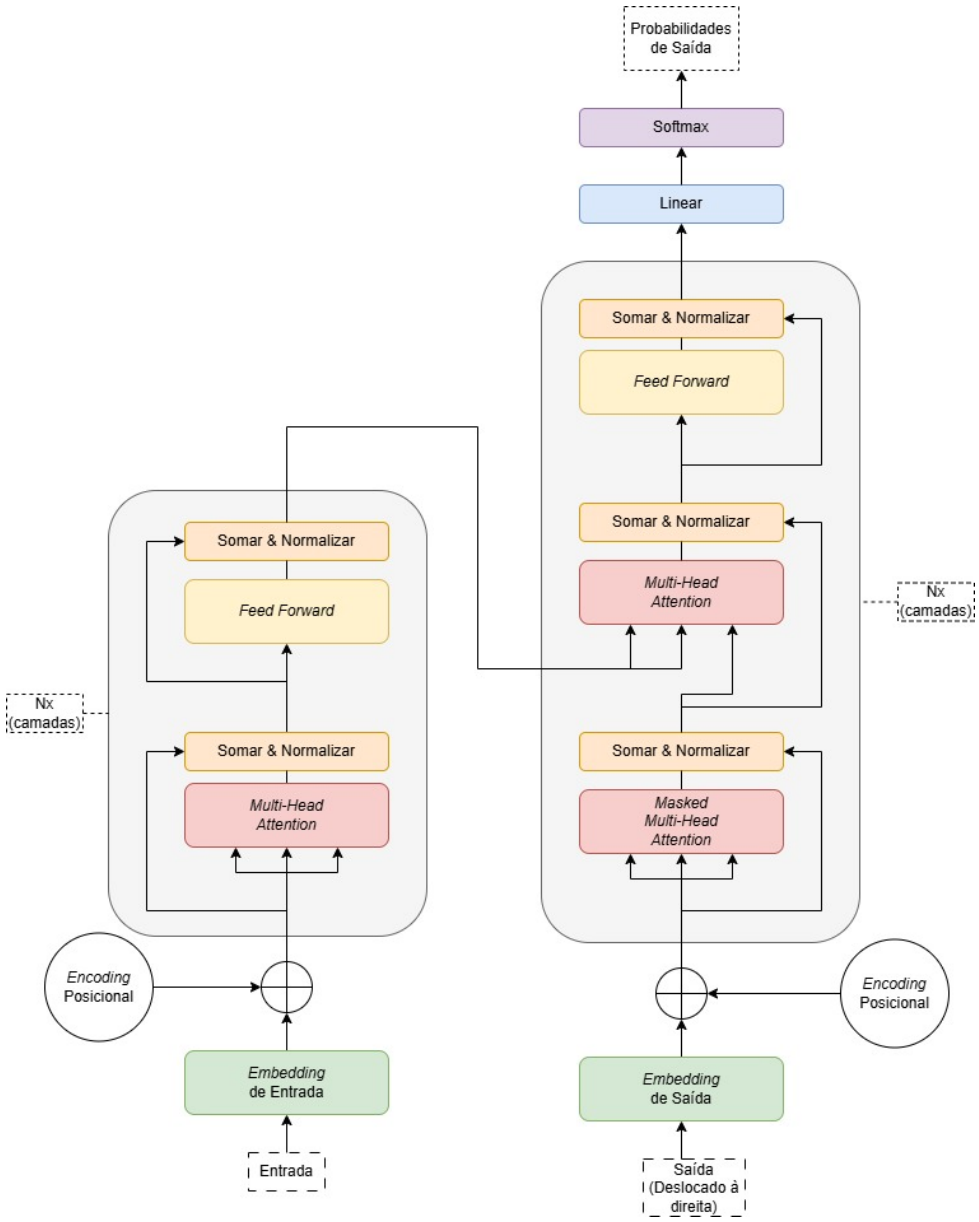

Figura 2.4: Arquitetura do modelo *transformer*.
Fonte: (VASWANI et al., 2017)

2.2.2.1 *Scaled Dot-Product Attention*

O mecanismo de *Self-Attention* do modelo *transformer* é chamado de *Scaled Dot-Product Attention* (Figura 2.5).

Segundo Vaswani et al. (2017), um mecanismo de atenção pode ser descrito como o mapeamento de uma consulta (*query*) e um conjunto de pares de chave (*key*) & valor (*value*)

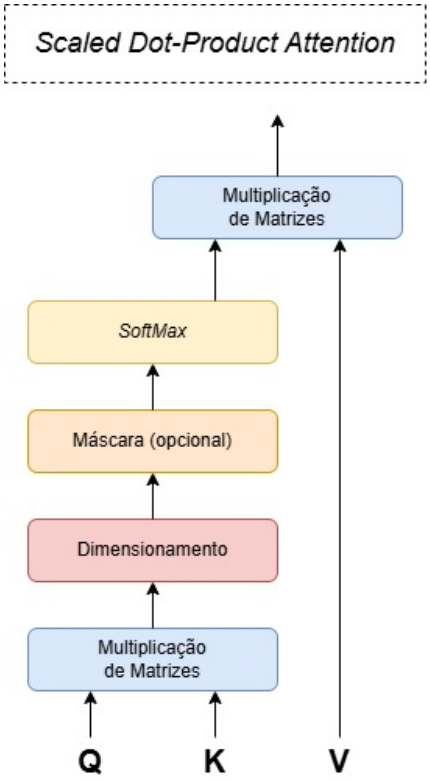

Figura 2.5: Mecanismo de atenção.
Fonte: (VASWANI et al., 2017)

a uma saída, onde a saída é computada como uma soma ponderada dos valores, e o peso atribuído a cada valor é calculado por uma função de compatibilidade da consulta com sua respectiva chave.

A atenção do modelo se dá pelas informações obtidas neste mapeamento de consulta, chave, e valor, onde as relações entre as palavras são calculadas para se criar uma representação vetorial que incorpora o contexto da sentença.

Este mecanismo de atenção utiliza três matrizes de entrada Q (*queries*), K (*keys*), e V "*values*". O nome *Scaled Dot-Product Attention* se origina pelas duas operações principais realizadas para calcular a atenção, o "*Dot-Product*", onde se faz a multiplicação entre as matrizes Q e K para obter o mapeamento das palavras, e "*Scaled*", onde se faz o dimensionamento da matriz resultante da multiplicação por uma constante $\sqrt{d_k}$ (d_k = dimensão da saída), para impedir que os valores resultantes da multiplicação se tornem muito grandes. Depois destes passos é realizada a função *Softmax* para normalizar os valores entre o espaço de 0 a 1, e a multiplicação pela matriz V para obter os pesos dos valores.

Este processo pode ser resumido em uma única fórmula que realiza as operações descritas simultaneamente:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Este cálculo pode ser dividido em h partes paralelas para permitir melhor aprendizado do contexto da sentença pela máquina. Estas partes então são concatenadas de volta para uma única saída. A esta execução em paralelo se dá o nome de *Multi-Head Attention* (Figura 2.6).

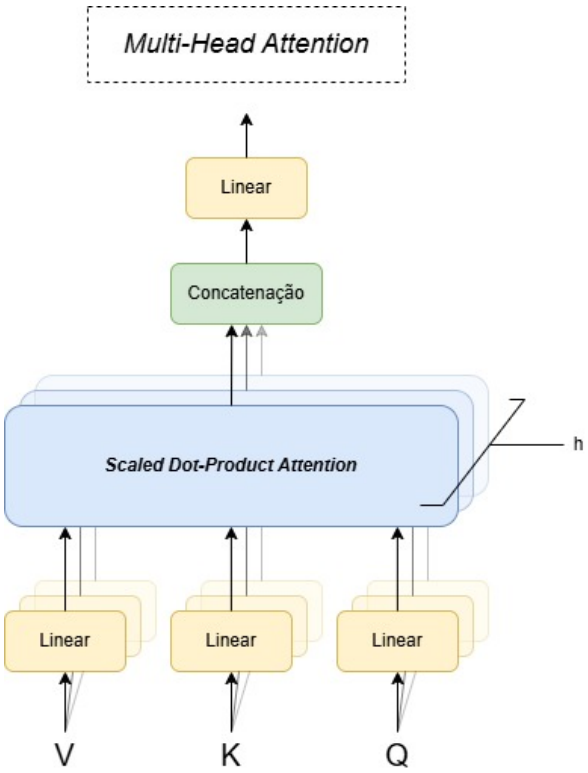

Figura 2.6: Representação da execução em paralelo *Multi-Head Attention* da arquitetura *transformer*.

Fonte: (VASWANI et al., 2017)

2.2.2.2 Encoder

A representação do *encoder* é desenhada na Figura 2.7, e as letras em parênteses na figura correspondem às explicações das partes a seguir. O *encoder* recebe apenas uma entrada, a sentença convertida em *embedding* e somada a uma codificação posicional (*Positional Encoding*) para indicar a posição de cada palavra na sentença. Em seguida, entra no componente de aprendizado profundo que é composto por duas subcamadas.

1. A primeira subcamada é o mecanismo *Multi-Head Self-Attention* do *encoder* (a). Este componente tem o papel de descobrir o significado de cada palavra e a relação com as demais.

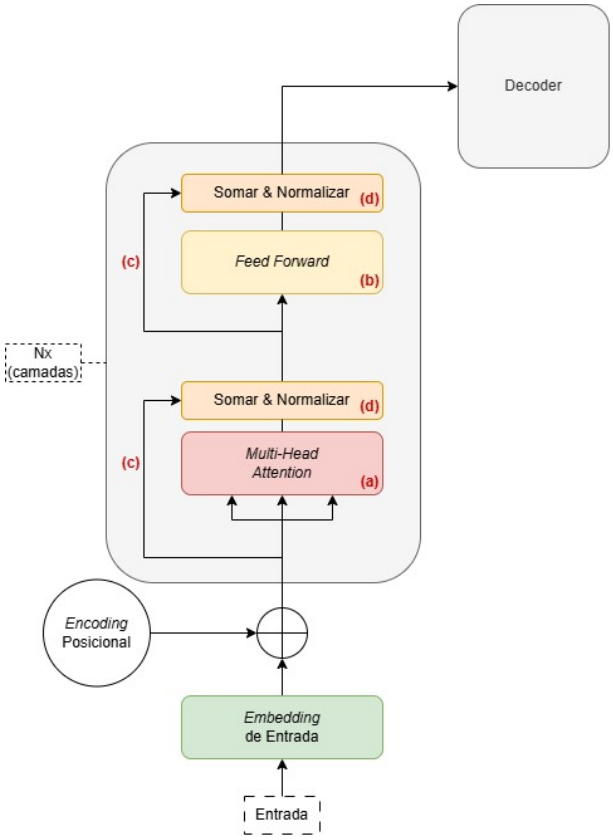

Figura 2.7: Desenho do *encoder* da arquitetura *transformer*.
Fonte: (VASWANI et al., 2017)

2. A segunda subcamada é o *Feed-Forward Network* (b), para ajustar os pesos das atenções pela rede neural.

Após estas duas subcamadas está presente um componente de conexão residual (c) e normalização (d). Esta conexão residual é realizada para impedir que o modelo perca o contexto original da sentença, antes que o modelo prossiga com as transformações seguintes. Para isto, a matriz de saída da subcamada atual (a saída do mecanismo de atenção, ou da rede neural) é somada com uma cópia da matriz de entrada, no estado anterior das transformações da subcamada. Em seguida, é feita uma normalização dos valores resultantes da soma de matrizes, com o intuito de reduzir o tempo de processamento em operações futuras.

A representação resultante pode então ser usada pelo *decoder* para tarefas de PLN.

2.2.2.3 Decoder

O *decoder*, desenhado na Figura 2.8, recebe duas entradas. A primeira entrada é a sentença convertida em *embedding* e somada ao *Positional Encoding*. A segunda entrada do *decoder* é a

saída das representações aprendidas pelo *encoder* (a). O *decoder* é composto por três subcamadas.

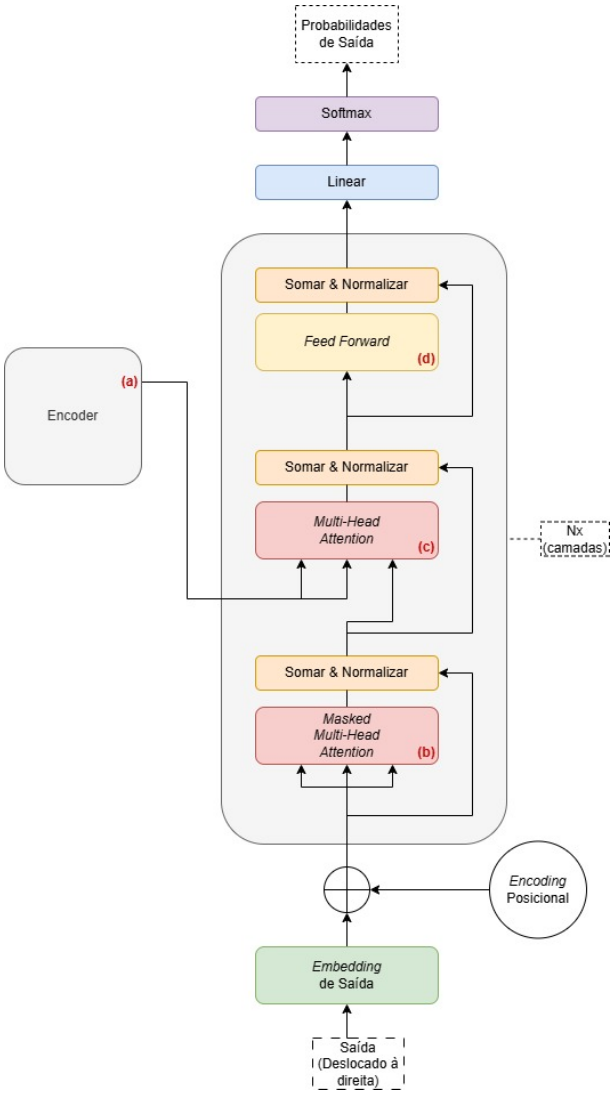

Figura 2.8: Desenho do *decoder* da arquitetura *transformer*.
Fonte: (VASWANI et al., 2017)

1. A primeira subcamada é o mecanismo *Masked Multi-Head Self-Attention* (b). Este mecanismo de atenção possui uma máscara de dados que previne a máquina de aprender as informações dos *tokens* futuros. Esta máscara é a substituição de *tokens* ainda não vistos da matriz resultante de QK^T pelo valor $-\infty$, para que na fase de normalização seja convertida em 0. Esta subcamada faz com que a máquina aprenda a interpretar apenas os tokens da sequência já vistos, e futuramente predizer o próximo *token* de acordo com estas informações.

2. A segunda subcamada é o mecanismo *Multi-Head Attention* (c). Desta vez, a matriz de saída do *encoder* (a) é incluída como as entradas K e V deste mecanismo de atenção, enquanto

a entrada Q é a mesma matriz de saída da subcamada anterior. Isto faz com que o mecanismo deixe de ser auto-atenção, e passe a considerar o que foi previamente aprendido pelo *encoder*.

3. Assim como no *encoder*, a última subcamada é a *Feed-Forward Network* (d), novamente realizando os ajustes nos pesos da rede neural.

Após cada subcamada, assim como no *encoder*, é feita a soma da conexão residual e normalização, para que a máquina não perca o contexto inicial.

O *decoder* apresenta comportamentos diferentes durante o treinamento e a inferência. Durante o treinamento, a matriz de *embedding* de entrda é movida um *token* à direita (*Shifted Right*) para permitir a inserção de um *token* especial chamada de *Start Of Sentence (SOS)* que indica o início da predição de palavras. Isto resulta em uma sentença de entrada no formato: [SOS] + "sentença alvo de predição"+ [EOS]. No caso de Inferência, a matriz de *embedding* de entrada é substituída apenas pelo token [SOS]. A máquina busca encontrar qual a melhor palavra para se colocar no *token* seguinte, e repete o processo de decodificação até alcançar o *token* de fim de sentença *End Of Sentence (EOS)*.

2.2.3 BERT

Bidirectional Encoder Representations from Transformers (BERT) é um modelo publicado em 2018, e é uma implementação bidirecional do modelo *transformer* (DEVLIN et al., 2018).

A forma como o modelo *transformer* tradicional aprende o contexto de uma sentença é considerada unidirecional, pois o modelo aprende a predizer cada palavra em um único sentido (leitura da esquerda para direita, de línguas como português e inglês), e aprende apenas o contexto das palavras vistas anteriormente na sequência de predições. A arquitetura tradicional, por ser unidirecional, não é capaz de entender completamente o contexto de sentenças em que as informações mais importantes estão no final da sentença. Intuitivamente, aprender o contexto de uma sentença bidirecionalmente (tanto da esquerda para direita quanto direita para esquerda) permite que o modelo aprenda mais detalhes sobre o contexto. Entretanto, ter a visão de todas as palavras ao mesmo tempo pode impactar a predição de palavras seguintes, pois facilita esta predição e reduz a capacidade de generalização do modelo.

A principal evolução do BERT é a capacidade de aprender o contexto de uma sentença bidirecionalmente, ao aplicar duas tarefas não-supervisionadas durante o pré-treinamento: *Masked Language Model (MLM)* e *Next Sentence Prediction (NSP)*.

2.2.3.1 Representação de entrada do modelo BERT

A Figura 2.9 ilustra o processamento das sentenças de entrada para o modelo BERT.

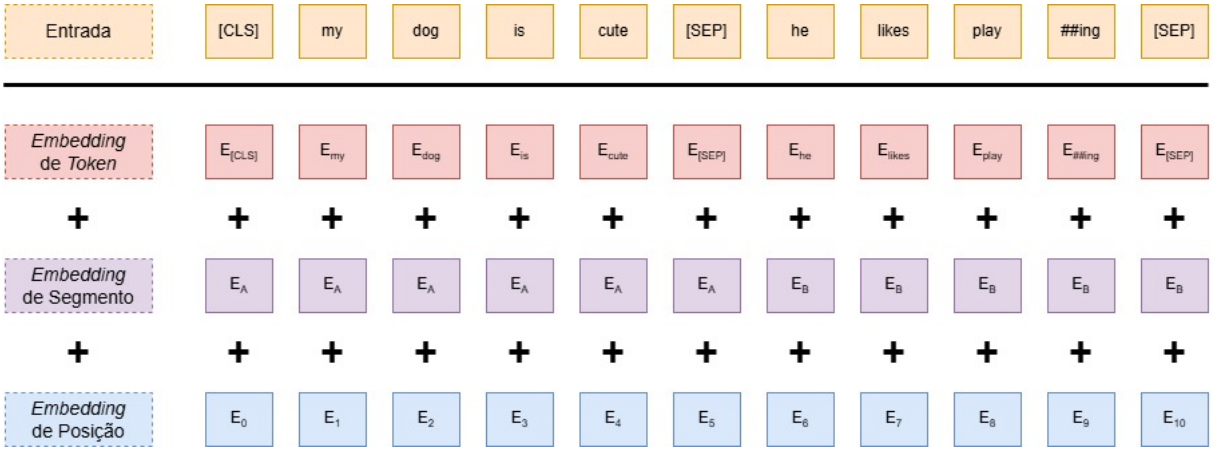

Figura 2.9: Representação das sentenças de entrada para o modelo BERT.
Fonte: (DEVLIN et al., 2018)

O BERT é capaz de receber uma única sentença ou um par de sentenças como entrada, de acordo com a tarefa de PLN desejada. O modelo *transformer* original utiliza como entrada os *embeddings* de *tokens* somados aos *embeddings* de posição. Os mesmos componentes podem ser encontrados no modelo BERT nos retângulos vermelhos e azuis respectivamente, na Figura 2.9. O BERT possui uma representação adicional, chamada de *embeddings* de segmentos, que indicam em qual sentença palavra pertence, o que permite o uso de funções como MLM e NSP.

Além disso, antes de qualquer sentença de entrada, é adicionado um *token* especial ([CLS]). Durante o cálculo de atenção, o *token* [CLS] que está no início da sentença é multiplicado pelos *embeddings* de todas as palavras, e isto permite ao [CLS] representar toda a sentença.

O modelo BERT é treinado de duas formas: O Pré-treinamento e o *Fine-Tuning*. Ambos passos utilizam da mesma arquitetura, sendo a única diferença a alteração da camada de saída. O pré-treinamento consiste em treinar o modelo em uma grande quantidade de dados, para que o modelo tenha conhecimento da língua em uma forma geral. Já o *Fine-Tuning* é o refinamento do modelo já pré-treinado para um contexto específico, e posteriormente utilizado em tarefas de PLN.

2.2.3.2 Pré-treinamento

O pré-treinamento é um passo inicial para os LLMs aprenderem linguagem natural de uma forma abrangente. O pré-treinamento do BERT foi realizado com a coleção BooksCorpus (800 milhões de palavras) e English Wikipedia (2,5 bilhões de palavras) (DEVLIN et al., 2018).

O pré-treinamento do BERT é realizado em dois passos, *Masked Language Model* (MLM) e *Next Sentence Prediction* (NSP).

O treino de uma representação bidirecional sem impacto na capacidade de predição é permitido pelo MLM. Ao invés de deixar todos os *tokens* visíveis para treinamento, uma parte dos *tokens* é aleatoriamente mascarada pelo *token* [MASK], para serem preditos. Embora isto permita um modelo bidirecional, ao manter apenas o *token* [MASK] durante o treinamento, futuramente o modelo não saberá predizer qual palavra deve ser predita neste momento. Portanto, apenas 15% dos tokens da sequência são escolhidos para serem tratados pelo MLM. Dentre eles, em 80% das ocorrências os *tokens* são realmente substituídos pelo *token* especial [MASK], enquanto 10% dos *tokens* são substituídos por uma palavra aleatória, e os 10% restantes são mantidos como o *token* original. Desta forma, é possível utilizar a máscara para permitir o treinamento bidirecional, enquanto se mantém a capacidade de predizer o *token* esperado.

A *Next Sentence Prediction* é uma simples tarefa de predição binária com o objetivo de treinar o modelo para entender relações entre sentenças, para tarefas como perguntas e respostas (Q&A). Ao selecionar duas sentenças A e B para predição, 50% das vezes a sentença B é uma sentença que realmente segue A, rotuladas de $IsNext$ e 50% restantes são sentenças aleatórias rotuladas de $NotNext$.

A arquitetura do BERT é baseada nos *encoders* da arquitetura *transformer*, e pode ser representada por três parâmetros principais: O número de camadas (L), o tamanho do *embedding* (H) e número de camadas em paralelo, ou *attention heads* (A). O BERT foi introduzido com o treinamento em dois tamanhos, o $\text{BERT}_{\text{BASE}}$ com 110 milhões de parâmetros (L=12, H=768, A=12), e $\text{BERT}_{\text{LARGE}}$ com 340 milhões de parâmetros (L=24, H=1024, A=16) (DEVLIN et al., 2018).

2.2.3.3 *Fine-Tuning*

O *fine-Tuning* é a tarefa de transferir o conhecimento geral de língua adquirido durante o pré-treinamento para propósitos mais específicos. Isto é realizado ao fornecer ao modelo novos

dados de um domínio mais específico, seguido por um treinamento de menor escala para que o modelo absorva os detalhes deste domínio.

O *Fine-Tuning* do BERT utiliza o mesmo modelo do pré-treinamento, sendo necessário apenas adaptar as sentenças de entrada e a camada de saída.

Para tarefas como paráfrase, perguntas e respostas e relação de premissa & hipótese, a entrada do BERT se torna um par de sentenças (A,B), concatenadas por meio de um *token* especial de separação ([SEP]). Em tarefas de classificação de texto e rotulação de sequências, a entrada do BERT se torna uma única sentença combinada a um elemento vazio (A,∅).

2.2.3.4 *Inferência*

Para a inferência, as representações de *tokens* são alimentadas em uma camada de saída para tarefas como Q&A, enquanto a representação [CLS] é alimentada à camada de saída para tarefas de classificação.

O estado final do *token* [CLS] é utilizado em tarefas de classificação ao comparar o *token* com o rótulo da classe esperada. Isto é possível pois [CLS] é um *token* que representa toda a sentença, e o *Fine-Tuning* para classificação aprende a aproximar o valor de [CLS] para os valores dos rótulos alvos da classificação.

2.2.4 *Large Language Models*

Large Language Models (LLM) é um nome dado aos modelos de aprendizado profundo treinados em grandes conjuntos de dados, geralmente baseados nas estruturas de *encoder* e *decoder* da arquitetura *transformer*.

Os modelos baseados em *encoder* se destacam em tarefas focadas na leitura e entendimento do texto, como classificação, análise de sentimento, detecção de entidades nomeadas, entre outros, pois não são capazes de gerar texto. Um exemplo de LLM baseado em *encoder* é o BERT, mencionado na Seção 2.2.3.

Os modelos baseados em *decoder* são focados na geração de texto, por meio da geração repetida de palavras mais prováveis de serem verdadeiras dado um texto inicial (*prompt*). O Chat-GPT é um LLM desenvolvido pela OpenAI e disponível publicamente, e recebe grande atenção pela sua aplicação em diversas áreas (RAY, 2023). Este modelo também foi desenvolvido sobre a arquitetura *transformer*, mas, ao contrário do BERT, a arquitetura permanece unidirecional (RADFORD et al., 2018).

A partir do ano de 2023 começaram a surgir trabalhos que utilizam Chat-GPT para auxiliar na geração de análise STPA. Estes trabalhos serão detalhados na Seção 2.3.2.

2.2.5 Classificação de texto

Classificação, em aprendizado de máquina, consiste em predizer uma classe (ou rótulo) de uma determinada entrada.

Existem algoritmos baseados em lógica, como Árvore de decisão, algoritmos estatísticos, como Naïve Bayes, algoritmos baseados em instâncias, como K-Vizinhos mais Próximos, entre outros (KOTSIANTIS, S. B.; ZAHARAKIS; PINTELAS et al., 2007).

Algoritmos de classificação tradicionais geralmente utilizam números como entrada, tanto inteiros quanto reais. Dados quantitativos permanecem em sua forma numérica, enquanto dados qualitativos como texto (em um domínio finito) geralmente são convertidos em uma escala de números inteiros.

A classificação de texto, ao contrário de valores tradicionais, contém dados majoritariamente não-numéricos. Como visto em tarefas de PLN, a classificação de texto requer que as palavras sejam representadas de forma numérica para que seja possível alimentar a máquina.

Com o auxílio de técnicas de PLN, é possível realizar a classificação de texto utilizando algoritmos tradicionais, após a conversão dos dados textuais em representações numéricas (*embeddings*). No entanto, o surgimento de algoritmos de aprendizado profundo, assim como *transformers* (VASWANI et al., 2017), elevou ainda mais o estado da arte, e métricas para entendimento de língua natural, como o teste *General Language Understanding Evaluation (GLUE)* (WANG et al., 2019), começaram a surgir.

A partir de 2018, com a popularização de modelos baseados na arquitetura *transformer* como o BERT (DEVLIN et al., 2018) e GPT (RADFORD et al., 2018), os LLMs começaram a ser cada vez mais utilizados para a tarefa de classificação de texto.

Os LLMs utilizados para classificação de texto podem variar em diversos aspectos: modalidade (tipo de dado a ser classificado, como apenas texto ou múltiplos tipos como texto, vídeo, número e áudio), números de parâmetros treinados, o número máximo de *tokens* permitidos como entrada, segurança e transparência dos dados do modelo (FIELDS; CHOVANEC; MADIRAJU, 2024).

De acordo com a pesquisa realizada em (FIELDS; CHOVANEC; MADIRAJU, 2024), o modelo BERT (ou modelos derivados de BERT) apresentou melhores resultados para *datasets* de diversas tarefas relacionada à classificação, como análise de sentimentos, classificação de texto multirrótulo, pergunta e resposta e classificação de sentenças.

A classificação por meio do modelo BERT, que será utilizada neste trabalho, requer apenas ajustar a última camada para camada de *output* para realizar a classificação (DEVLIN et al., 2018), pois utiliza o *token* especial [CLS] como a representação da sentença como um todo (SUN et al., 2019). Durante o *fine-tuning*, o *token* [CLS] é comparado com a saída (classe) esperada, e os pesos de atenção são ajustados de acordo com o erro calculado. Já na inferência, a sentença de entrada passa pelo modelo BERT com o *fine-tuning* já realizado, e o valor do *token* [CLS] indica a probabilidade da sentença ser rotulada com a determinada classe.

Algumas pesquisas recentes utilizam modelos classificadores baseados em *transformers* dentro do contexto de engenharia de requisitos e segurança.

Varenov e Gabdrahmanov (2021) utilizam três modelos baseados em *transformers* (BERT, XLNET, e distil-BERT) para classificar requisitos de segurança em 7 classes: "confidencialidade", "integridade", "disponibilidade", "transparência", "operacional", "controle de acesso", e "outros". Os autores compilaram um *dataset* de diversas fontes que resultou no total de 1086 sentenças de classes desbalanceadas. Para o *dataset* compilado, o distil-BERT obteve o melhor F1-Score de 78,00% após a remoção da classe "outros", por ser uma classe com características muito abrangentes ou que dependem de conhecimento humano especializado. Os autores argumentam que a natureza reduzida do distil-BERT contribui para obter melhores resultados neste *dataset*, pois permite o uso de menos dados de treinamento para o modelo aprender sobre os dados.

Feng et al. (2021) utilizam um modelo composto por *embedding* de BERT em conjunto com BiLSTM + Atenção para extração de atributos para classificar textos em 5 níveis de severidade dos eventos analisados pela técnica HAZOP (*Hazard and Operability Analysis*). Os autores obtiveram um *dataset* de 2075 textos de análise de uma empresa da indústria petroquímica. O modelo combinado atingiu o resultado mais alto de 88,20% de F1-Score. Os autores observaram que o modelo foi capaz de absorver conhecimento de segurança, como eventos relacionados à resíduos e corrosões de materiais pertencendo à severidades mais baixas, enquanto eventos relacionados à explosões, fogo e poluição ambiental pertencendo à severidades mais altas.

Estes conhecimentos obtidos pela máquina são próximos do que um analista humano entende de severidade de cada tipo de evento.

2.2.6 Similaridade de sentenças

Uma das tarefas realizadas em PLN é o cálculo de similaridade entre sentenças.

Similaridade Textual Semântica (do inglês *Semantic Textual Similarity*) "avalia o grau em que duas sentenças são semanticamente equivalentes entre si" (CER et al., 2017). Tarefas derivadas do cálculo de similaridade de sentenças incluem o reconhecimento de implicação textual, relacionamento semântico e detecção de paráfrases, mas estes se diferem da similaridade textual por serem classificações binárias de relacionamento entre sentenças ao invés de um cálculo gradativo de sobreposições dos significados entre as sentenças (CER et al., 2017).

Assim como discutido em 2.2.1, é necessário inicialmente converter os textos em representações numéricas (*embeddings*) para transformar o texto em uma entrada reconhecível à máquina. Após gerar representações de duas sentenças, é feito o cálculo da distância entre os vetores numéricos por meio de funções de distância.

A biblioteca *sentence-transformers* é baseada na arquitetura *Sentence-BERT* (REIMERS; GUREVYCH, 2019), uma rede siamesa que combina dois modelos BERT e que foi treinada para tarefas que envolvem múltiplas sentenças. Ela se destaca por ser capaz de criar *embeddings* que representam a sentença inteira.

O BERT realiza a concatenação entre duas sentenças e calcula a similaridade com base em cada *token* da sentença, o que requer maior tempo de processamento para realizar a comparação entre múltiplas sentenças. *Sentence-BERT* se sobressai em relação ao BERT em tarefas de similaridade de sentenças por calcular apenas um único vetor de tamanho fixo para representar cada sentença. Isto reduz o poder computacional necessário para calcular a similaridade entre sentenças, pois a máquina realiza a comparação apenas com os vetores já existentes, ao invés de calcular vetores toda vez para cada par de sentenças. Em um experimento de agrupamento de 10,000 sentenças, o *Sentence-BERT* foi capaz de reduzir para 5 segundos a inferência que levou ao BERT original 65 horas para ser computada.

A função de distância utilizada neste trabalho é a similaridade de cosseno da biblioteca *sentence-transformers*.

De acordo com a documentação do *Framework* de aprendizado profundo PyTorch (Pytorch documentation), utilizado neste trabalho, o cálculo da similaridade entre duas sentenças se dá pela seguinte fórmula:

$$Similarity = \frac{x_1 \cdot x_2}{max(||x_1||_2, \epsilon) \cdot max(||x_2||_2, \epsilon)}$$

Onde x_1 e x_2 são os vetores representantes das sentenças, $||x_1||$ e $||x_2||$ são os mesmos vetores normalizados, e ϵ um valor extremamente pequeno para evitar a divisão por zero. Esta operação resulta em um valor real entre [0,1], onde 0 significa que dois vetores são diferentes, e 1 significa que os dois vetores são idênticos.

A vantagem desta função é que, por se comparar o valor de dois vetores normalizados, a função permite a comparação entre dois elementos de tamanhos distintos.

2.3 Trabalhos relacionados

O levantamento de estudos relacionados foi realizado por meio do método *Preferred reporting items for systematic reviews and meta-analyses*, PRISMA (MOHER et al., 2010) para uma revisão sistemática, em conjunto com o levantamento de outros artigos selecionados arbitrariamente.

Inicialmente foi realizado um levantamento de artigos de forma arbitrária pelo autor, com o intuito de entender o domínio do problema. Este levantamento foi realizado no mecanismo de busca Google Acadêmico , devido à facilidade de uso e abrangência dos resultados provenientes de diversas bases de artigos. Os termos de busca foram uma combinação de termos como "*Machine Learning*", "*Classification*", "*Requirements*", "*Critical Systems*", e "STPA".

Em seguida, iniciou-se a revisão sistemática de literatura com base no método PRISMA, de forma adaptada para atender às necessidades da pesquisa. As bases bibliográficas utilizadas para a revisão foram ACM Digital Library, IEEE Xplore, Science Direct e Web of Science, bases de alto impacto, da área de tecnologia da informação ou multidisciplinares. O espaço temporal de busca foi definido em um período de 6 anos mais recentes (janeiro de 2018 a 2024), e os termos de busca utilizados foram [("machine learning" OR "classification" OR "LLM" OR "BERT" or "transformer") AND "STPA"], com o intuito de encontrar artigos que utilizam LLM, classificação, BERT, *transformer* ou aprendizado de máquina em geral, como soluções de tarefas de STPA.

Os resultados foram filtrados para selecionar apenas os artigos em revistas ou conferências que contêm os termos mencionados anteriormente, com um filtro por título e resumo nos mecanismos de busca. Não foi utilizado o filtro diretamente por título, pois STPA é um domínio pequeno e o filtro limita o escopo de busca.

A seleção de artigos relevantes foi feita por meio da remoção de artigos duplicados, e a remoção por irrelevância de resumo.

Este levantamento resultou em 59 resultados iniciais provenientes das quatro bases bibliográficas. Este número foi reduzido para 46 artigos ao eliminar 13 resultados duplicados, depois reduzido a 11 resultados ao descartar 35 títulos ou resumos não relevantes.

A Figura 2.10 ilustra a revisão sistemática realizada.

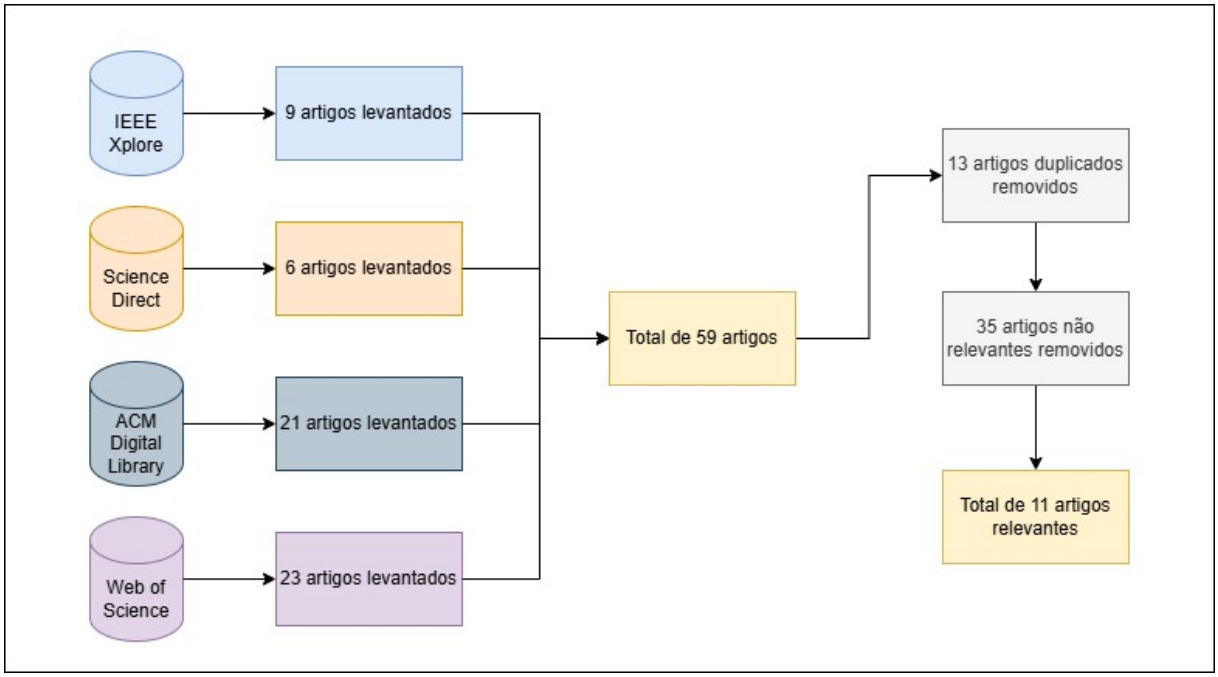

Figura 2.10: Resultados obtidos pelo método PRISMA.

Após este levantamento, foi possível concluir que não existem trabalhos relacionados ao tema desta pesquisa.

Entre os 11 artigos resultantes um deles é um mapeamento de literatura sobre STAMP. Todos os 10 artigos restantes, no entanto, não estão diretamente relacionados aos objetivos da pesquisa. Embora estes artigos contenham os termos "machine learning"e "STPA" no título ou resumo, esta combinação nos artigos se refere à aplicação da técnica STPA para análise de perigo em sistemas de aprendizado de máquina existentes, como componentes de veículos autônomos. Isto se torna contrário ao objetivo da pesquisa, em que se busca utilizar

tecnologias de aprendizado de máquina para auxiliar em tarefas de STPA. Alguns destes artigos são apresentados nas seções seguintes.

2.3.1 Trabalhos relacionados à Análise STPA e aprendizado de máquina

Acar Celik et al. (2022) aplicam STPA em um componente de detecção de colisão de pedestres em carros autônomos, que utiliza aprendizado de máquina. Houve uma preocupação em aplicar STPA neste componente, pois normas em veículos automotivos como ISO 26262 não descrevem explicitamente como regulamentar algoritmos de aprendizado profundo.

Assim como (ACAR CELIK et al., 2022), o trabalho de Ghosh et al. (2023) trata de componentes de carros autônomos. Neste trabalho, os autores aplicam STPA para identificar riscos de ciberataques em componentes de software, e os componentes de aprendizado de máquina em foco são algoritmos de captura de informação externa, como radares, por exemplo. Os autores argumentam que, com o aumento de carros autônomos em circulação nos últimos anos, também há o aumento de possíveis ataques a estes veículos. Embora existam guias de análise de ameaças para veículos como o padrão ISO 21434 mencionado pelos autores, ainda não existem métodos desenhados especificamente para carros autônomos. Portanto, os autores realizaram uma comparação entre o padrão ISO 21434 e STPA-Sec, uma adaptação da STPA para segurança. As vantagens da aplicação da STPA-Sec, levantada pelos autores, são a capacidade de modelar interações maliciosas na via, como objetos e outros usuários, e identificar caminhos críticos para o funcionamento do veículo.

Em (RISMANI et al., 2023), os autores exploram o uso de aprendizado de máquina responsável e os riscos sociais e éticos de quando um modelo de aprendizado de máquina se comporta de forma inadequada. Neste trabalho, os autores realizaram entrevistas com 30 participantes, de diferentes cargos e experiências, para avaliar as práticas responsáveis de pesquisa ou engenharia de software em relação ao uso de modelos de aprendizado de máquina. Eles consideram estes modelos genericamente, de diversos domínios, como componentes que geram dados ou predições. Já a definição de riscos sociais e éticos foi uma pergunta realizada aos entrevistados, em que cada um explicou seu entendimento, e concordam que não há uma definição específica na comunidade. No entanto, há um consenso entre os entrevistados que os riscos sociais e éticos são violações aos direitos humanos que podem surgir de uma avaliação de impacto aos direitos humanos, e podem ser interpretados

como uma entrada ou saída indevida de um modelo de aprendizado de máquina. Como uma tentativa de melhorar a segurança nestes modelos, os autores sugerem o uso de técnicas de análise de perigo (STPA e FMEA) para identificar perigos. A técnica STPA foi elogiada no quesito de que atende bem às necessidades dos stakeholders (usuários ou público alvo), e também ajuda a mapear relações entre os humanos e os sistemas de aprendizado de máquina. Entretanto, é apontado que são necessárias uma colaboração e organização mais fortes entre os desenvolvedores e stakeholders. Além disso, é necessária a inclusão de um especialista no time, tanto social quanto de segurança, pois ao envolver entidades e técnicas externas, o projeto se torna multidisciplinar.

2.3.2 Trabalhos relacionados à Análise STPA e LLM

Até o momento, existem apenas três trabalhos que utilizam LLMs para auxiliar a análise STPA. Estes trabalhos são descritos abaixo.

Diemert e Weber (DIEMERT; WEBER, 2023) investigam a integração de LLMs nos processos de análise de Perigos, a qual os autores chamam de *Co-Hazard Analysis (CoHA)*. No método CoHA, os autores ensinam o Chat-GPT 3.0 uma codificação para entender a descrição do sistema de interesse e fornecem ao modelo diversas consultas para identificar as Ações de Controle não seguras do sistema descrito. A resposta do LLM é então marcada em três categorias: a resposta é correta e útil, a resposta está correta mas não útil, e a resposta está incorreta. Os autores então comparam a proporção de cada marcação em diferentes níveis de complexidade de sistema. Os autores afirmam que CoHA pode ser moderadamente útil para sistemas mais simples. No entanto, quanto maior a complexidade do sistema, o desempenho do LLM decai.

Em (QI et al., 2023), são realizados três experimentos utilizando Chat-GPT 3.5 para a geração de uma análise STPA sobre um sistema de freio de emergência automático, ou Automated Emergency Braking (AEB). No experimento denominado *One-off Simplex Interaction*, o time de especialistas fornece ao Chat-GPT as palavras chaves "AEB System" e "STPA Method", sem fornecer detalhes de como realizar STPA. É considerado *One-off* pela interação ocorrer apenas uma vez no início do experimento, e Simplex pela interação ser unidirecional, ou seja, ocorre apenas o fornecimento das palavras chaves. No experimento denominado *Recurring Simplex Interaction*, a informação é fornecida múltiplas vezes (*recurring*) pela direção do time de especialistas para o Chat-GPT, mas ainda se mantém

unidirecional. Por fim, no experimento denominado *Recurring Duplex Interaction*, ocorrem interações recorrentes entre o time de especialistas e Chat-GPT bidirecionalmente, onde os autores fornecem informação ao LLM, recebem a resposta do modelo, e retornam correções ou observações ao modelo para aprimorar respostas futuras. Os autores concluem que o Chat-GPT pode apresentar instabilidade devido à geração automatizada, como não trazer todos os resultados esperados, dificuldade de reprodução dos resultados e propagação de erros quanto maior a conversa. No entanto, quanto maior o envolvimento humano, o LLM apresenta melhores resultados.

Charalampidou et al. (CHARALAMPIDOU; ZELESKIDIS; DOKAS, 2024) utilizam LLMs em seus experimentos para comparar o tempo necessário para realizar a análise STPA com e sem a ajuda do Chat-GPT 4.0. Os autores utilizaram um método de 5 passos para interagir com o LLM: fornecem informações básicas, como o sistema de interesse e seus limites, as Ações de Controle não seguras, cenários de perda e especificações de segurança, para que o modelo seja capaz de reornar os passos 3 e 4 da análise STPA. Os autores concluem que o Chat-GPT não deve ser usado isoladamente para gerar análise STPA, pois metade das respostas obtidas não estavam em um estado satisfatório e exigiam validação por especialistas de STPA. No entanto, Chat-GPT pode ser útil para analistas experientes como uma ferramenta de *brainstorming* e reduzir o tempo necessário para a análise, já que o LLM foi capaz de gerar algumas Ações de Controle não seguras que não foram identificadas inicialmente por analistas.

Todos estes três trabalhos recomendam não usar LLMs como substitutos de analistas humanos, mas sim como uma ferramenta de auxílio para reduzir o tempo requerido para completar a análise.

Em uma participação no Workshop "*Latin American STAMP Workshop*" 2024 (OKAMURA et al., 2024), o autor desta dissertação apresentou a utilização de aprendizado de máquina para auxiliar a análise STPA. Nesta apresentação, foram mostrados a criação de um *dataset* de sentenças utilizadas durante o primeiro passo da análise STPA, e o uso de algoritmos de classificação de aprendizado de máquina tradicionais (*Support Vector Machine* e *Naïve Bayes*).

Dois experimentos foram realizados, com cada um dos classificadores: i) a classificação entre as classes de Perda, Perigo e Restrição de um conjunto do *dataset* que contém apenas sentenças escritas corretamente, de acordo com o guia de STPA, e ii) a classificação das mesmas classes, mas utilizando um conjunto de dados que inclui sentenças possivelmente escritas de forma incorreta. Foi concluído que os classificadores têm mais facilidade em

classificar o conjunto de sentenças corretas, comparado ao conjunto misto de sentenças corretas e incorretas. Uma versão atualizada deste *dataset* será discutida na Seção 3.2.

Com base neste levantamento bibliográfico, podemos entender que a aplicação de aprendizado de máquina em tarefas de análise de perigo STPA ainda é um tópico pouco estudado e é uma lacuna no conhecimento ainda a ser explorada.

Este trabalho se difere da literatura existente ao usar modelos baseados em *encoders* da arquitetura *transformer*, e usar uma combinação de modelos especializados e treinados em diferentes dados para classificar erros em uma sentença do primeiro passo da análise STPA. Ou seja, ao invés de utilizar LLMs para gerar análise STPA, que pode introduzir tempo e esforço para verificar o texto gerado por máquina, este trabalho propõe a abordagem de usar LLMs para auxiliar os analistas em sua análise STPA ao fornecer ideias de possíveis erros e correções, e aumentar sua produtividade.

No capítulo a seguir será abordada a metodologia da pesquisa.

Capítulo 3

Metodologia

A Figura 3.1 mostra a visão geral da metodologia de desenvolvimento. A metodologia é composta por duas partes principais. A primeira parte (Seção 3.2) se refere à criação de um *dataset* que contém sentenças textuais geradas durante o primeiro passo da análise STPA (Perdas, Perigos em Nível de Sistema e Restrições em Nível de Sistema), representada pelo retângulo vermelho na Figura 3.1. Os retângulos de cor cinza representam subconjuntos deste *dataset*, particionado para uso em passos futuros da metodologia. A segunda parte (Seção 3.3), representada pelos retângulos amarelos e laranja na Figura 3.1, se refere à criação do pipeline de aprendizado de máquina para o treinamento dos modelos classificadores. A esse pipeline foi dado o nome de BEDS (BERT Error Detection for STPA). O retângulo azul se refere aos testes do pipeline a serem discutidos no Capítulo 4.

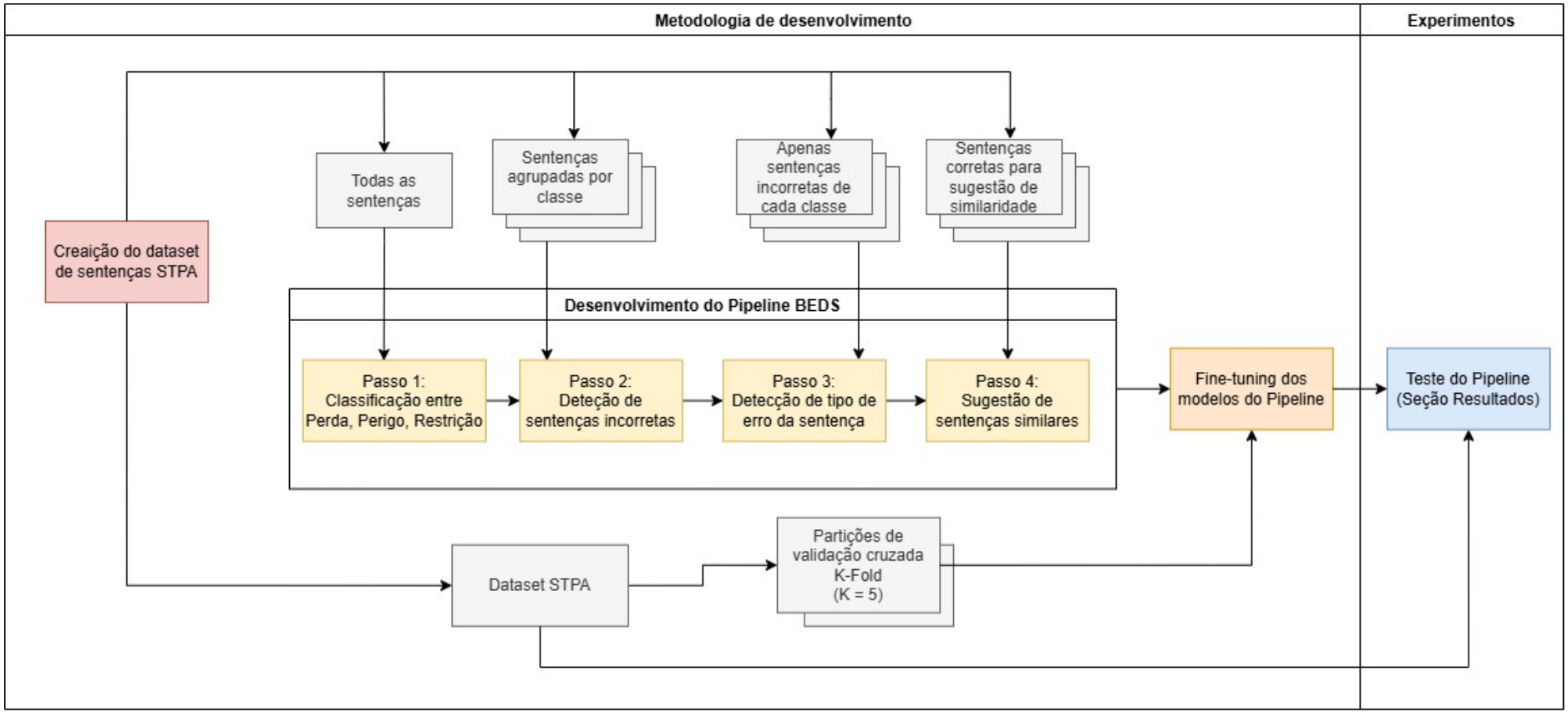

Figura 3.1: Visão geral da metodologia.

3.1 Ambiente de desenvolvimento e experimentação

Os experimentos foram realizados na plataforma em nuvem *Google Colab* com a assinatura *Pro* em uma GPU A100, devido à facilidade de acesso.

As principais bibliotecas usadas foram: *pandas 2.2.2*, para importação de arquivos e manipulaçõa de *dataset*; *scikit-learn 1.6.1*, para preparação de dados em tarefas de ML; *pytorch 2.5.1*, um framework para modelos de aprendizado profundo; *transformers 4.45.2*, para o uso de modelos baseados em arquitetura *transformer*; e *sentence-transformer 3.1.1*, para treinamento e uso do modelo de similaridade de sentenças.

Durante os três primeiros passos do pipeline, o mesmo modelo baseado em *transformer* foi utilizado (BERT-base-uncased), mas refinado com diferentes conjuntos de dados para capturar o contexto necessário em cada passo. No quarto passo o modelo "all-mpnet-base-v2" foi utilizado, pois é o modelo mais baixado para tarefas de similaridade de sentenças de acordo com o repositório HuggingFace (Hugging Face Repository).

3.2 Criação do *Dataset*

O *dataset* de sentenças é um componente indispensável para esta pesquisa, pois elas são utilizadas como dados de treinamento para os modelos BERT, e posteriormente como fonte de sugestões de sentenças corretas. No entanto, durante o levantamento de trabalhos relacionados foi feita uma busca por dados relacionados à análise STPA, e não foram encontrados *datasets* que envolvem esta técnica. Isto incentivou a criação de um *dataset* de sentenças dedicado a este Pipeline, mas que inclui diversos metadados para serem reaproveitados em pesquisas futuras.

Todos os dados e código-fonte dos programas desenvolvidos neste trabalho estão disponíveis neste repositório[1].

Para a criação deste *dataset*, foram extraídas sentenças de valores identificados durante o primeiro passo da análise STPA, de apresentações realizadas entre 2012 a 2023 e disponíveis publicamente na página de apresentações do *MIT STAMP Workshop* (MIT STAMP Workshop).

As sentenças foram extraídas de listas ou tabelas que contêm exemplos de análise de um determinado sistema ou domínio da apresentação. Estas sentenças foram registradas no *dataset* junto com o rótulo da tabela extraída ("*loss*", "*hazard*", ou "*constraint*"), metadados

[1] https://github.com/andreyokamura-unicamp/BEDS-Pipeline

relacionados à apresentação para permitir a rastreabilidade, e rótulos de classificação adicionados para treinamento do pipeline. O nome e conteúdo de cada atributo é explicado na Tabela 3.1.

Tabela 3.1: Arquitetura do *Dataset*.

Índice	Coluna	Explicação
1	"sentence" (sentença)	A sentença extraída da apresentação.
2	"label" (rótulo)	A classe da sentença (Perda, Perigo ou Restrição).
3	"validation" (validação)	Indica se a sentença é correta ou incorreta.
4	"error" (erro)	Indica o tipo de erro da sentença incorreta.
5	"domain" (domínio)	Domínio da apresentação.
6	"year" (ano)	Ano da apresentação.
7	"title" (título)	Título da apresentação.
8	"source" (fonte)	URL da apresentação.
9	"slide" (slide)	O número do slide em que a sentença foi extraída.

Uma versão anterior deste *dataset* foi apresentada no "*Latin American STAMP Workshop*" (OKAMURA et al., 2024). Na versão atual, duas novas colunas foram adicionadas: a coluna "*validation*"que indica se uma sentença está escrita de forma correta ou incorreta, e a coluna "*error*"que indica o tipo de erro da sentença, caso incorreta. Além disso, este *dataset* foi verificado por especialistas em análise STPA para permitir o treinamento dos modelos do pipeline em rotulações mais confiáveis. Os critérios de rotulação serão detalhados abaixo.

Todos os atributos contidos neste *dataset* podem ser encontrados na apresentação de onde cada sentença foi extraída (como a classe relacionada, nome da apresentação, URL de acesso, entre outros metadados). Apenas duas colunas com informações externas foram adicionadas: os atributos "*validation*" e "*error*".

O atributo "*validation*" permite dois valores "*correct*", para sentenças consideradas corretas, e "*incorrect*", para sentenças consideradas incorretas. O *STPA Handbook* (LEVESON, N.; THOMAS, 2018) fornece recomendações de como realizar a análise STPA e identificar os principais valores do sistema, estas instruções serviram de base para a decisão do rótulo de cada sentença. Por exemplo: sentenças sobre Perigos em Nível de Sistema devem conter um "sistema" de interesse e uma "condição não segura", as quais em uma situação de ambiente de pior caso podem levar a uma Perda; sentenças sobre Restrições em Nível de Sistema devem conter um "sistema" e uma "condição segura a ser cumprida", de uma

forma que seja oposta aos Perigos, ou mesmo "o que deve ser feito para minimizar Perdas" caso um "Perigo" ocorra; e sentenças sobre Perdas, que não são dados um formato específico, mas são descritas como um valor ou "*stake*" identificado no sistema pelos *stakeholders*, e convertidos em Perdas (com o uso de palavras como "Loss of", or "Damage to" + valor).

No entanto, podem existir sentenças que não estão explicitamente escritas de acordo com as recomendações fornecidas pelo *STPA Handbook* (LEVESON, N.; THOMAS, 2018). Isto significa que as sentenças podem conter informações desnecessárias para sua análise, falta de contexto, ou possuem significados ambíguos a depender do contexto inserido. Nestes casos, as sentenças devem ser rotuladas como "*incorrect*". Este é um dos desafios do primeiro passo da análise STPA, pois as sentenças que contém estas características podem ser frequentemente consideradas corretas.

Para o atributo "*error*" são atribuídos valores apenas quando as sentenças são consideradas incorretas, o que significa dizer que o atributo "*validation*" recebeu o valor incorreto.

Para o atributo "*error*", o autor dessa pesquisa analisou as sentenças e as classificou de acordo com quatro erros:

i) "é um acidente"; ii) "não pertence à classe original", iii) "necessita reescrita"; e iv) "sentença contém condições". Detalhes de cada rótulo podem ser vistos na Tabela 3.2.

Os valores do atributo "*validation*" foram verificados por especialistas em análise STPA para garantir que as sentenças devem realmente ser rotuladas como corretas ou incorretas. Dois especialistas de STPA contribuíram com a verificação do *dataset*, ambos com 3 a 4 anos de experiência profissional na indústria aeronáutica. A verificação da coluna em questão foi realizada em 3 passos. Inicialmente, foi feita uma reunião entre o autor e os especialistas para discutir as regras da análise STPA em conjunto com as características das sentenças da análise, e entrar em um acordo para decidir quais características tornam uma sentença correta ou incorreta. Em seguida, com base nas regras discutidas, o autor marcou cada uma das sentenças do *dataset* com os rótulos "*correct*"ou "*incorrect*"de acordo com as características da sentença. Por fim, os especialistas verificaram a rotulação realizada pelo autor, para confirmar se o rótulo atribuído está em consenso com a sua decisão à respeito da sentença. Para sentenças em que houve diferença de valores entre o autor e os especialistas, foi dada a prioridade de decisão para a interpretação do especialista. Isto garante que dados confiáveis sejam utilizados para o treinamento dos modelos do Pipeline.

Tabela 3.2: Tipos de erro das sentenças incorretas do *dataset*.

Tipo	Classe envolvida	Raciocínio
accident (é um acidente)	Perda, Perigo	Um acidente é uma situação que certamente levará à Perda, mas não é necessariamente um *stake* para os *stakeholders*. Acidentes também diferem de Perigos, pois Perigos são condições de sistema que em situações de ambiente de pior caso podem levar a Perdas, enquanto acidentes são situações resultantes de Perigos que levam diretamente às Perdas.
not (não pertence à classe original)	Perda, Perigo, Restrição	Sentenças que podem mudar de significado de acordo com o contexto (por exemplo, o que é considerado um sistema ou um *stake*), e é difícil de decidir a qual classe pertence. Quaisquer outras sentenças que não pertencem à classe original e não podem ser categorizadas entre os erros existentes são incluídas.
rewrite (necessita de reescrita)	Perda, Perigo, Restrição	Sentenças sem alguma palavra chave (como o *stake*, o sistema, condição segura ou não segura) para serem consideradas corretas. Algumas sentenças podem conter toda informação necessária, mas as palavras precisam ser reordenadas para transmitir a informação corretamente. Sentenças que necessitam de pequenas correções para serem consideradas corretas são inclusas.
condition (sentença inclui condições)	Perdas	Sentenças que contêm palavras como "[Perda] due to [...]", "[Perda] from [...]", ou "[Perda] caused by [...]" dão à sentença um significado de causa efeito ou situações que dependem de condições, e podem limitar o escopo da análise, impedindo de encontrar outras situações que podem levar às Perdas. O mesmo pode ser dito a sentenças que contêm "[Perda] during [...]" ou "[Perda] while [...]" que limitam as Perdas apenas para situações específicas, que não é o ideal.
	Perigos	Sentenças que contêm palavras como "[Perigo] causes [...]" ou "[Perigo] results in [...]" dão à sentença de causa e efeito, e provavelmente resulta em um acidente ou Perda. Um Perigo descreve um estado ou condição do sistema que em situações de ambiente de pior caso podem levar à Perdas, mas nos exemplos acima, a sentença já define quais são os resultados deste Perigo. Isto pode dificultar a ligação de demais Perdas e Perigos.

Durante a extração das sentenças, foram encontradas apresentações sem nenhuma sentença a ser extraída ou que continham exemplos parciais da análise (por exemplo, uma tabela contendo apenas sentenças de Perdas e Perigos, mas sem Restrições), o que resultou em um número desbalanceado de sentenças para cada classe. O *dataset* contém o total de 1.034 sentenças, e a distribuição de classes pode ser vista na Tabela 3.3.

Tabela 3.3: Distribuição de casses do *dataset*.

Classe	Nº de sentenças
Perda	291
Perigo	424
Restrição	319
Total	1034

Das 1034 sentenças no *dataset*, 31,33% (324) de todas as sentenças provêm do domínio da aeronáutica, seguido pelo domínio automobilístico com 15,66% (162) das sentenças, e o domínio da medicina com 9,67% (100) das sentenças. Os três maiores domínios são os únicos domínios com o número acima de 100 sentenças, e eles representam 56,67% de todo o *dataset*. A distribuição de domínios pode ser vista na Tabela 3.4. Nesta tabela, apenas os domínios com um número acima de 20 sentenças foram incluídos, enquanto os demais domínios foram agrupados na categoria "outros".

Durante a criação dos modelos de classificação foi percebido um desbalanceamento de classes para o atributo "*validation*", o que impactava negativamente o processo de treinamento e inferência dos modelos classificadores. A Tabela 3.5 mostra a proporção de sentenças corretas e incorretas para cada classe.

Enquanto as classes Perdas e Perigos possuíam em torno de 33% das sentenças rotuladas como incorretas, a classe Restrições continha apenas 18 (5,64%) sentenças incorretas. Este desbalanceamento causa situações em que não existem exemplos suficientes para o treinamento do modelo, notadamente após dividir o *dataset* entre conjuntos de treinamento, validação e teste. Para permitir a continuação dos experimentos foi necessário aumentar o número de sentenças de Restrições incorretas. Portanto, foram adicionadas 50 sentenças parafraseadas e criadas com base nos exemplos de Restrições incorretas existentes no *dataset*. Estas sentenças foram criadas por meio de alterações no domínio, no sistema de interesse, na

Tabela 3.4: Distribuição de domínios do *dataset*.

Domínio	Nº de sentenças
aeronáutica	324
automotiva	162
medicina	100
aeroespacial	90
nuclear	54
ferroviária	52
marítima	45
laboratório	39
militar	37
indústria	36
outros	95

Tabela 3.5: Distribuição de classes entre os atributos "*label*" e "*validation*".

Classe	Sentenças corretas	Sentenças incorretas	% de sentenças incoretas
Perda	192	99	34,02%
Perigo	287	137	32,31%
Restrição	**301**	**18 (68)**	**05,64% (18,42%)**

condição a ser cumprida ou na ordem de palavas das sentenças existentes. Esta adição resulta em um *dataset* com um total de 1.084 sentenças, e a proporção de sentenças de restrições incorretas foi aumentada para 18,42%. Os valores após o incremento são apresentados em parênteses na Figura 3.5.

3.3 Proposta: Pipeline de detecção de erros com BERT para análise STPA (BEDS)

3.3.1 Arquitetura do Pipeline

O Pipeline BEDS (BERT Error Detection for STPA) é composto por quatro passos. O fluxo de execução é descrito pelo diagrama de atividades na Figura 3.2.

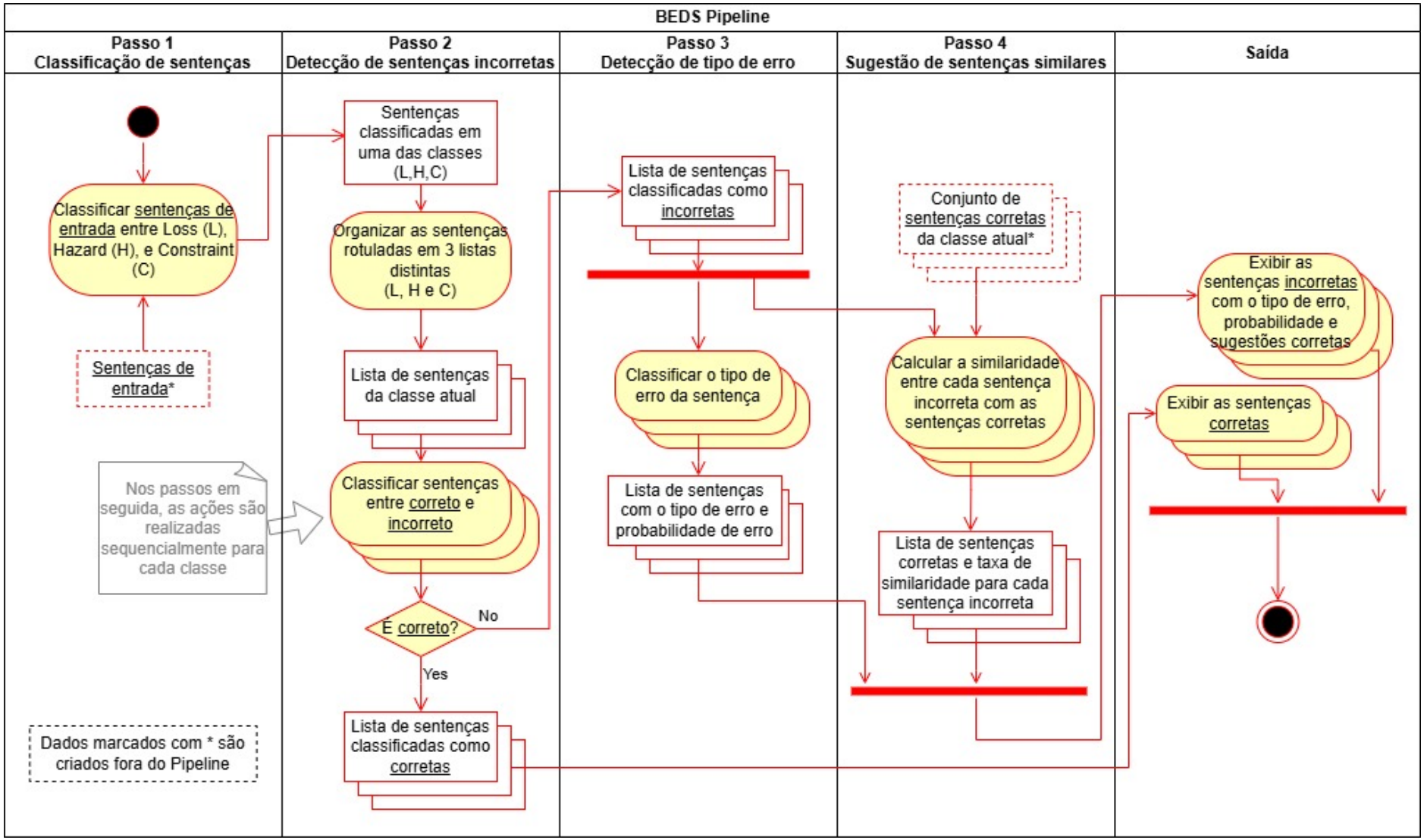

Figura 3.2: Diagrama de atividades do Pipeline BEDS.

3.3.2 Passo 1: Classificação das sentenças em Perdas, Perigos ou Restrições

O primeiro passo do Pipeline, ilustrado na primeira raia à esquerda na Figura 3.2, consiste em apenas um modelo classificador (atividade amarela). Este classificador espera uma lista (ou um *dataset* de única coluna convertida em lista) que contém apenas sentenças textuais relacionadas ao primeiro passo da análise STPA e retorna as predições, organizadas em um *dataset* com a sentença e o rótulo predito (entre "*loss*", "*hazard*", e "*constraint*").

Este passo pode ser pulado caso o usuário já possua um *dataset* com sentenças rotuladas (colunas “sentence” e “label”), ou utilizado intencionalmente ao remover a coluna de rótulos para entrada do modelo. O propósito deste passo é utilizar o modelo de classificação para classificar sentenças em sua classe adequada (Perda, Perigo ou Restrição) dadas as características da sentença, independentemente se a sentença contém erros ou não. Estes erros serão descobertos pelo segundo e terceiro passos deste Pipeline.

Erros de classificação ao utilizar um *dataset* já rotulado como entrada no passo 1 (por exemplo, uma sentença com rótulo original de Perda ser predita como Perigo) servirão como um detector primário de ambiguidade. Modelos de aprendizado de máquina aprendem características (como o uso de certas palavras e contexto) de cada classe vista durante o treinamento. Erros de classificação indicam que a sentença classificada possui características vistas com mais frequência em outra classe e não apresenta as características esperadas de uma sentença da classe originária, o que indica uma possível ambiguidade.

Este é o único passo manual do pipeline, onde o programa requer que o usuário insira as sentenças de entrada que deseja verificar. A classificação e as atividades dos passos seguintes são realizadas automaticamente, com base nos dados inseridos inicialmente.

3.3.3 Passo 2: Detecção de sentenças incorretas

O segundo passo é ilustrado pela segunda raia à esquerda na Figura 3.2, e consiste de três modelos de classificação diferentes, um para cada classe (Perdas, Perigos e Restrições). Este passo espera um *dataset* com duas colunas ("*sentence*"e "*label*") como entrada. Este *dataset* é então organizado em três listas de sentenças separadas por sua classe de acordo com o rótulo atribuído na coluna "*label*", e em seguida cada lista é inserida em seu respectivo classificador.

Deste passo em diante, todas as ações são realizadas paralelamente para cada classe (Perdas, Perigos e Restrições).

Este passo retorna predições para cada classe, que indicam se a sentença inserida é considerada correta ou incorreta. As sentenças consideradas incorretas são agrupadas em listas e utilizadas como entrada para o passo seguinte. O passo 2 representa a função principal deste Pipeline, que é determinar se a sentença está correta ou incorreta.

3.3.4 Passo 3: Detecção de tipo de erro da sentença

O terceiro passo também consiste de três modelos de classificação, como ilustrado pela raia central da Figura 3.2. Este passo espera três listas de sentenças incorretas geradas no passo anterior. Cada classe possui um conjunto de possíveis erros (definidos na Tabela 3.2), e os classificadores são utilizados para identificar qual erro está presente na sentença.

As classificações resultantes são organizadas em dicionários que contêm a sentença analisada, o tipo de erro predito, e as probabilidades para cada tipo de erro. Um símbolo de ramificação é posicionado na raia central para representar o uso da mesma entrada (uma lista de sentenças incorretas) tanto no passo 3 quanto no passo 4 do Pipeline.

Nos passos 2 e 3 do Pipeline existe um modelo classificador para cada classe (Perda, Perigo e Restrição). Esta separação foi feita para permitir que o modelo aprenda características e palavras específicas de cada classe, sem a necessidade de gastar recursos para aprender a generalizar e descobrir como diferenciar entre sentenças de cada classe (o que já foi feito no passo 1).

3.3.5 Passo 4: Sugestão de sentenças similares

O quarto e último passo, ilustrado na quarta raia na Figura 3.2, utiliza um modelo de similaridade de sentenças para sugerir sentenças classificadas como corretas que são mais parecidas com as saídas geradas pelo passo 2.

O passo 4 retorna um dicionário que contém a sentença incorreta analisada, e n sentenças mais similares a esta sentença em conjunto com a porcentagem de similaridade. Esta porcentagem é calculada por meio da função de similaridade de cosseno entre duas sentenças: a sentença incorreta, e toda lista de sentenças corretas. O resultado é então ordenado para selecionar as n sentenças mais similares. O propósito deste passo é fornecer sugestões para corrigir a sentença. Como são apenas sugestões, estas sentenças devem ser

consideradas apenas como referências para adaptar e corrigir a sentença incorreta, e não são a decisão final de correção.

Para o cálculo de similaridade, este passo utiliza três listas externas que contêm apenas sentenças corretas, criadas a partir do *dataset* descrito na Seção 3.2. Estas três listas são criadas em dois passos: Primeiro, é criado um subconjunto do *dataset* que contém apenas as sentenças rotuladas como correta na coluna "*validation*"; em seguida, as sentenças de cada classe (Perda, Perigo e Restrição) são agrupadas em suas respectivas listas. Desta forma, apenas as sentenças corretas de cada classe podem ser utilizadas como referência para o cálculo de similaridade.

3.3.6 Exibição dos resultados

Na raia à direita da Figura 3.2, é ilustrada a finalização do Pipeline após a exibição ao usuário das sentenças corretas, e para cada sentença incorreta, o tipo de erro detectado pelos classificadores, a probabilidade de cada erro, e a lista de sentenças similares.

A presença de sentenças em todas as classes não é obrigatória, então o Pipeline é capaz de executar cada classe independentemente.

3.3.7 *Fine-tuning* dos modelos

A Figura 3.3 ilustra com mais detalhes as colunas utilizadas no treinamento de cada modelo de classificação, e os rótulos que cada modelo aprende a classificar.

Este Pipeline utiliza apenas as colunas "*sentence*", "*label*", "*validation*", e "error" presentes no *dataset* criado na Seção 3.2 (representado pelo retângulo cinza na Figura 3.3). Para a avaliação dos modelos, foi utilizada a técnica de validação cruzada *K-Fold* ($K = 5$), estratificados na coluna "*label*". Isso representa 5 iterações de treinamento e teste com diferentes conjuntos de dados para cada iteração (4/5, ou 80% para treinamento, e 1/5, ou 20% para teste). Os dados utilizados para teste dos modelos não foram empregados em nenhum momento durante o treinamento, para todos os classificadores. A média entre os resultados das 5 iterações foi utilizada como o desempenho final de cada passo do Pipeline, e o modelo da iteração com a maior métrica F1 foi salva no repositório 1 para uso futuro em teste de execução. Todos os rótulos para classificação foram convertidos em números inteiros (de 0 a n) de acordo com o número n de rótulos presentes em cada atributo do *dataset*, para permitir o funcionamento dos modelos durante o treinamento.

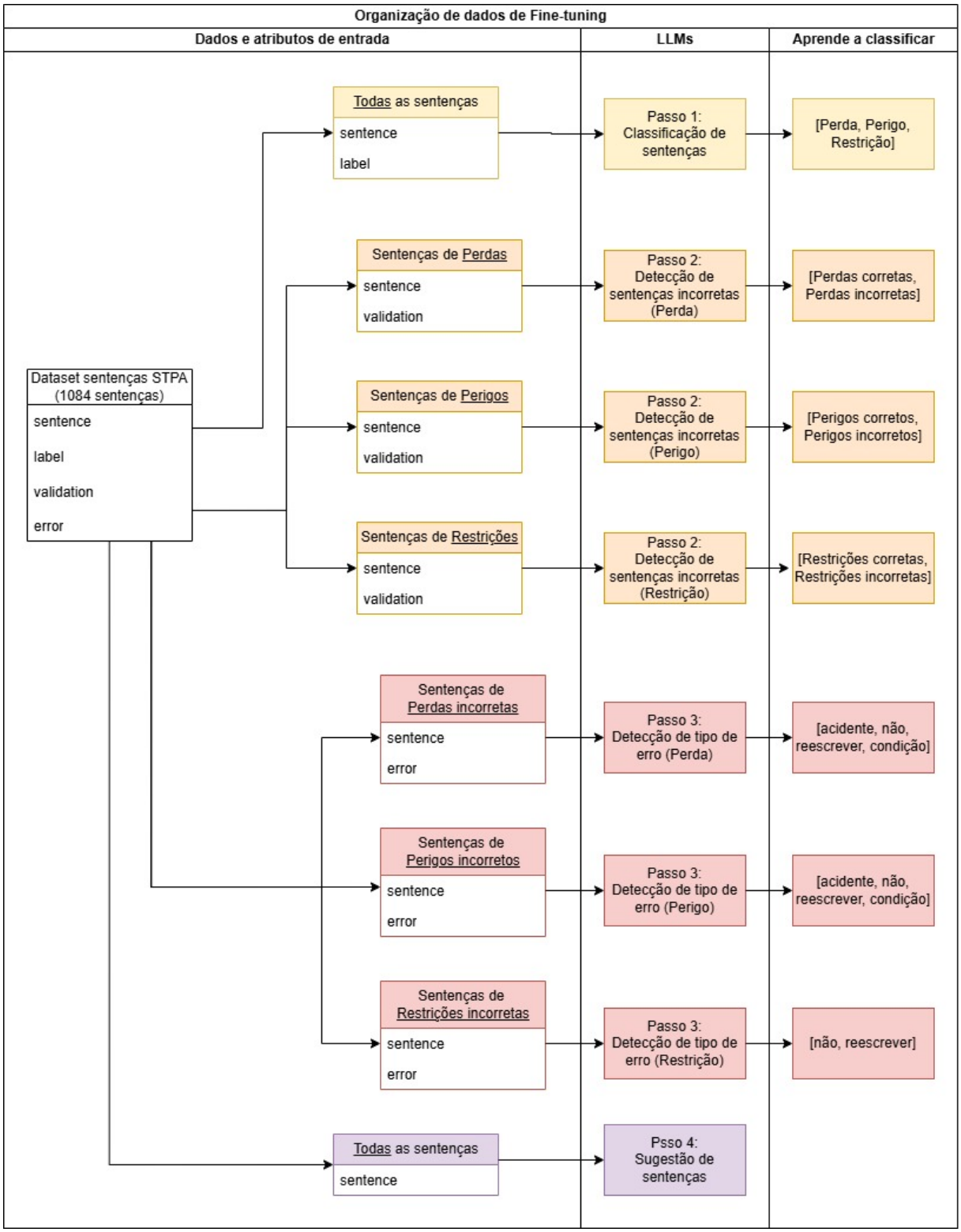

Figura 3.3: Os atributos de entrada esperados e os rótulos classificados de cada LLM do Pipeline.

O modelo de classificação no primeiro passo (representado pelos retângulos amarelos na Figura 3.3) foi treinado para detectar a classe de uma sentença (entre Perda, Perigo e Restrição), e o conjunto de treinamento contém as colunas "*sentence*" como dado e "*label*" como rótulo de treinamento.

No segundo passo, que é utilizado para detectar as sentenças corretas e incorretas, três modelos de classificação (representados pelas três linhas de retângulos laranjas na Figura 3.3) foram treinados. Cada um destes modelos foi treinado com um subconjunto do conjunto de treinamento que contém apenas os exemplos de sua respectiva classe (Perda, Perigo e Restrição). Este passo utiliza as colunas "*sentence*" como dado e "*validation*" como rótulo de treinamento.

No terceiro passo, que é utilizado para identificar o tipo de erro em uma sentença, três modelos de classificação (representados pelas três linhas de retângulos vermelhos na Figura 3.3) foram treinados. Desta vez, apenas exemplos incorretos de cada classe foram utilizados, o que resulta em um número reduzido de dados para treinamento. Este passo utiliza as colunas "*sentence*" como dado e "*error*" como rótulo de treinamento.

No quarto passo, um modelo de similaridade de sentenças baseado na arquitetura *sentence-transformer* foi treinado, como representado pelos retângulos lilases da Figura 3.3.

Este modelo foi treinado com o conjunto de treinamento inteiro para aprender características generalizadas sobre a análise STPA. Este modelo utiliza a função de perda “ContrastiveTensionLossInBatchNegatives”, em que a entrada esperada é um par de sentenças ("âncora", "âncora"), ou seja, a mesma sentença é utilizada como referência. Nesta função, os pares positivos e negativos que indicam se uma sentença é similar ou não durante o treinamento foram gerados automaticamente pelo algoritmo. Isto permite que o modelo aprenda características das sentenças sem a necessidade de supervisão, ou a criação de um *dataset* específico para treinar modelos de similaridade.

Os modelos de classificação treinados neste Pipeline foram avaliados com métricas comumente utilizadas em tarefas de classificação de aprendizado de máquina: acurácia, precisão, sensibilidade, e macro F1-score (IKONOMAKIS; KOTSIANTIS, S.; TAMPAKAS et al., 2005). Os tempos de treinamento e execução de cada passo também foram registrados.

O código de implementação do Pipeline pode ser encontrado no mesmo repositório do *dataset*1.

Capítulo 4

Resultados e Discussão

Os resultados da execução do Pipeline para os passos 1 a 3 com o conjunto de dados de teste podem ser encontrados na Tabela 4.1. Os melhores resultados de cada passo estão destacados em negrito. Esta tabela contém a média entre as 5 partições da validação cruzada *K-Fold*, das métricas de classificação (acurácia, precisão, sensibilidade e F1) e o tempo de processamento para treinamento e inferência do modelo (assume-se que os modelos já estão carregados). Para o cálculo de tempo de execução, foi utilizado o *dataset* inteiro.

Tabela 4.1: Resultados de execução dos classificadores do Pipeline.

Passo	Acurácia (σ)	Precisão	Sensibilidade	F1-Score (σ)	Tempo treino	Tempo teste
1	**95,20% (±0.94)**	**95,43%**	**94,91%**	**95,08% (±0.91)**	272,3s	233,9s
2 (Perda)	90,37% (±4.17)	90,24%	89,41%	89,08% (±4.78)	97,8s	64,7s
2 (Perigo)	79,00% (±3.96)	76,01%	75,98%	75,78% (±4.73)	124,2s	91,8s
2 (Restrição)	**96,47% (±1.83)**	**96,03%**	**92,40%**	**93,95% (±3.31)**	111,9s	79,6s
3 (Perda)	74,50% (±13.99)	67,41%	71,69%	66,35% (±15.26)	63,9s	22,3s
3 (Perigo)	79,59% (±5.71)	55,92%	60,14%	57,32% (±8.35)	66,9s	29,2s
3 (Restrição)	**96,25% (±5)**	**97,22%**	**95,41%**	**95,83% (±5.73)**	56,1s	15,1s

4.1 Resultados do Passo 1

No primeiro passo do Pipeline, o modelo classificador alcançou uma acurácia de 95,20% e F1-Score de 95,08%. Como este passo possui o maior número de sentenças utilizadas como entrada tanto para o treinamento quanto para teste, o tempo de processamento é o mais alto entre todos os modelos. Alguns erros de classificação foram destacados na Tabela 4.2. "Rótulo original" refere-se ao rótulo original extraído da apresentação, enquanto "Rótulo predito" refere-se ao rótulo detectado pelo modelo classificador para a sentença.

Tabela 4.2: Exemplos de erros de classificação do passo 1 do Pipeline.

Sentença	Rótulo original	Rótulo predito
(1) Severe braking.	Perigo	Perda
(2) Radiation Exposure.	Perda	Perigo
(3) The scientific data is rendered unusable (e.g., deleted, corrupted, not returned at required time) before it can be fully investigated.	Perda	Perigo

Os erros de classificação neste passo podem indicar que a sentença não está clara e é ambígua, ou foi rotulada incorretamente durante a análise. No entanto, os resultados indicam que os erros de classificação do primeiro passo ocorrem em sua maioria devido a ambiguidade de sentenças ao invés de erro de rotulação, e esta ambiguidade é encontrada em sentenças com certas palavras em comum, ou ordenação de palavras que permitem múltiplas interpretações, inclusive ser entendido como pertencente a outra classe.

Os erros de classificação mais frequentes ocorrem entre Perdas e Perigos. Embora o *STPA Handbook* (LEVESON, N.; THOMAS, 2018) forneça instruções para a escrita de Perigos (sentenças devem conter o sistema de interesse e uma condição não segura), podem existir casos em que o sistema está implícito, ou a condição não segura pode ser interpretada como uma perda para os *stakeholders*, o que causa uma ambiguidade entre as duas classes. Por outro lado, sentenças de Restrições são bem estruturadas e facilmente identificáveis devido à presença obrigatória de verbos modais (do inglês *modal verbs*, como "*must*" ou "*should*"), onde o substantivo (neste caso o sistema) deve preceder, e a condição a ser cumprida deve suceder o verbo modal, da seguinte forma: O "sistema" "deve" "cumprir uma condição segura".

Alguns exemplos da confusão entre sentenças de Perdas e Perigos (destacados na Tabela 4.2) são descritos abaixo.

No primeiro exemplo da Tabela 4.2 (1), a sentença "*Severe braking*" possui a classe original de Perigo (*Hazard*), mas o modelo detectou como uma Perda (*Loss*). De acordo com o *STPA Handbook* (LEVESON, N.; THOMAS, 2018), uma sentença de Perigo deve conter o sistema de análise e uma condição não segura, que em situação de ambiente de pior caso pode levar a uma Perda. Neste caso, a sentença "*Severe braking*" (frenagem brusca) pode ser considerada como uma condição não segura que leva à perda de vida ou perda de equipamento, mas a sentença não contém o sistema explícito a ser analisado (como o veículo ou motorista). A falta de sistema em uma sentença pode resultar na decisão de que a sentença é uma Perda. Outro raciocínio (e não necessariamente exclusivo) pode ser feito, em que o modelo entendeu "*Severe braking*" como uma forma próxima de "*Damage to vehicle*" (perda ou dano ao veículo), e consequentemente rotulou a sentença como Perda.

O segundo exemplo da Tabela 4.2 (2) apresenta o oposto do erro de classificação do exemplo anterior, em que uma Perda é classificada como um Perigo. Neste caso, a "*Radiation Exposure*" (exposição à radiação) pode ser interpretada não como uma Perda, mas sim uma possível fator causador de "*Loss of life*" ou "Loss of environment", que são exemplos mais comuns em sentenças de Perdas.

Da mesma forma o terceiro exemplo da Tabela 4.2 (3) apresenta o erro de uma sentença de Perda ser classificada como Perigo. Neste caso "*scientific data is rendered unusable*" teria o significado de *Loss of mission*, pois os dados de missão não podem mais ser usados. No entanto, o modelo pode ter interpretado "*scientific data*" como um sistema e "*is rendered unusable*" como uma condição não segura causadora de Perda, o que fez o modelo classificar a sentença como Perigo.

4.2 Resultados do Passo 2

No segundo passo, onde é feita a classificação de sentenças corretas e incorretas, os modelos atingiram a média entre os três classificadores de 88,51% para acurácia e 86,27% para F1-Score. A acurácia mais alta de 96,47% atingida pelo classificador de Restrições indica que os erros em sentenças de Restrições são mais fáceis de serem detectados. Por outro lado, o classificador de Perigos atingiu a menor acurácia de 79,00%, e apresenta maior dificuldade de classificar,

possivelmente devido a diversidade de formas que as sentenças de Perigo podem ser escritas. Alguns exemplos de erros de classificação são demonstrados na Tabela 4.3.

Tabela 4.3: Exemplos de erros de classificação do passo 2 do Pipeline.

Sentença	Rótulo original	Rótulo predito
(1) (Perda) Loss of integrity of communication.	Incorreto	Correto
(2) (Perigo) Loss of control or communications.	Incorreto	Correto
(3) (Perigo) Smartgrid has an inability to meet unexpected demands.	Correto	Incorreto

No primeiro exemplo, a sentença de Perda "Loss of integrity of communication" possui o rótulo original de sentença incorreta, mas o modelo classificou como correta. Embora a sentença contenha o trecho "Loss of" que é comumente esperado em sentenças de Perda, a perda de comunicação não é um valor dos *stakeholders*. A Perda de comunicação (com algum sistema) pode ser uma causa para perda de controle ou funcionamento do sistema, ou seja, se enquadra mais na classe de Perigos.

O segundo exemplo é uma sentença similar, mas com o rótulo original de Perigo incorreto sendo classificado correto. Pelo mesmo motivo anterior o modelo pode ter entendido a sentença como uma restrição correta, mas esta sentença não contém um sistema explícito (por exemplo uma aeronave) para que ela seja considerada correta.

Já o terceiro exemplo é uma sentença correta de Perigo, onde o *Smartgrid* (sistema) passa por uma inabilidade de atender demandas inesperadas (condição não segura). Não existe uma explicação clara para este erro de classificação, portanto existe uma possibilidade de ser um erro introduzido pelo próprio algoritmo durante o treinamento.

4.3 Resultados do Passo 3

No terceiro passo, os modelos identificam o tipo de erro contido nas sentenças incorretas descobertas pelo passo anterior. Os classificadores atingiram a média de 83,44% de acurácia e 73,16% de F1-Score. Diferentemente dos passos anteriores, existe uma lacuna significante entre os resultados atingidos, com uma diferença de 10,28% entre a média das duas métricas. Embora a acurácia indique que os modelos ainda são capazes de classificar os tipos de erro das sentenças, a medida F1 reduzida reflete a dificuldade dos modelos de diferenciar

múltiplos rótulos de tipos de erro (4 rótulos para Perdas e Perigos, e 2 rótulos para Restrições).

A Tabela 4.4 apresenta três resultados da execução do passo 3 do Pipeline (um exemplo para Perda, Perigo, e Restrição, respectivamente). Após a execução, este passo retorna um dicionário com três valores: a sentença incorreta analisada, o tipo de erro predito (em negrito na Tabela 4.4), e a probabilidade de cada tipo de erro.

Tabela 4.4: Exemplos de execução do passo 3 do Pipeline.

Sentença	Probabilidades
(1) "Aircraft crashes."	rewrite: 07,72% not: 03,83% condition: 07,36% **accident: 81,09%**
(2) "ACC did not maintain a safe distance from the object in the front, resulting in collision."	rewrite: 24,80% not: 14,12% **condition: 58,33%** accident : 02,75%
(3) "Avoid flight in altitude below permitted limit."	rewrite: 03,79% **not : 96,21%**

O primeiro exemplo ("*Aircraft crashes.*") se trata de uma sentença sobre acidentes de aeronaves. O modelo classificou a sentença corretamente com a probabilidade de 81,09% de ser "*accident*". Embora a sentença esteja errada de acordo com a STPA, a categoria acidente toma prioridade sobre as categorias "reescrever"ou "não pertence", pois acidentes são um tipo de erro frequente e possui uma classe específica para esse erro.

O segundo exemplo (*"ACC did not maintain a safe distance from the object in the front, resultng in collision."*) se trata do Controle de Cruzeiro Adaptativo (ACC) em veículos que não mantêm uma distância segura de objetos ao redor. Esta é uma sentença de Perigo que contém o sistema ("ACC") e a condição não segura, que em situação de pior caso, pode levar à Perda ("*did not maintain a safe distance from the object in the front*"). Esta sentença foi considerada incorreta, no entanto, devido à presença de condições de acontecimento que limitam o escopo da análise ("*resulting in collision*"). O tipo de erro foi devidamente identificado como "*condition*" com 58,33% de probabilidade.

O terceiro exemplo ("*Avoid flight in altitude below permitted limit.*") se trata de uma recomendação para manter uma altitude adequada para aeronave durante o voo. Embora a sentença pareça conter uma condição correta, não existe o sistema e nem um verbo que reforça a condição segura, como "*must*"ou "*should*", portanto a sentença foi classificada como "não" ser pertencente à classe de Restrições com 96,21% de probabilidade.

4.4 Resultados do Passo 4

O quarto e último passo do Pipeline utiliza um modelo de similaridade de sentenças para sugerir sentenças corretas a partir de um conjunto predeterminado de dados. Este passo pode sugerir n sentenças, número esse definido pelo usuário. A Tabela 4.5 apresenta um exemplo de execução do passo 4, com $n = 5$, como uma continuação dos resultados obtidos pelo passo 3 na Tabela 4.4. Este passo retorna dois valores: a sentença incorreta analisada, e a lista de n sentenças corretas que são mais similares a esta sentença, que podem ser utilizadas como referência para correção. As sentenças em negrito na Tabela 4.5 são as com o cálculo de similaridade de cosseno mais altas, e as sentenças sublinhadas indicam as sugestões que o autor considerou que são relevantes para a sentença incorreta analisada.

O primeiro exemplo se trata de um acidente que envolve uma aeronave. A característica principal necessária para a escrita de uma sentença de Perda correta é a identificação do objeto de valor aos *stakeholders* (neste caso a aeronave), e convertê-lo em uma Perda (Perda da aeronave). O modelo sugeriu sentenças similares que envolvem uma combinação de "*Loss of*", "*Damage to*", e "*Aircraft*". Embora a similaridade das sugestões seja relativamente baixa (50% para baixo), todas as sentenças sugeridas são relevantes e podem ser utilizadas para adequar a sentença incorreta.

O segundo exemplo trata do sistema de cruzeiro adaptativo que não manteve uma distância segura dos objetos ao redor. A sugestão com maior a similaridade (89,77%) é uma sentença exatamente igual a sentença analisada, com a diferença de que o trecho condicional ("*resulting in collision*") foi isolado entre parênteses, removido do contexto da sentença principal (e por isso esta versão foi considerada como correta). Apenas as duas primeiras sugestões foram consideradas relevantes para correção, pois envolvem o sistema ACC e a condição de distância.

O terceiro exemplo se trata de manter o voo (de um sistema omitido) em altitude adequada. A falta de um sistema explícito faz com que o contexto seja desconhecido,

Tabela 4.5: Exemplos de sugestões de sentenças do passo 4 do Pipeline.

Sentença	Similaridade	Sugestão
(1) "Aircraft crashes."	**51,31%**	**Loss of aircraft.**
	45,81%	Loss or damage to aircraft.
	45,79%	Damage to or loss of aircraft.
	43,44%	Loss of or damage to aircraft.
	42,97%	Damage to the aircraft.
(2) "ACC did not maintain a safe distance from the object in the front, resultng in collision."	**89,77%**	**ACC does not maintain a safe distance from the object in the front (resulting in a collision).**
	60,72%	ACC violates the safe distance between ACC vehicle and vehicle in front.
	51,83%	ACC did not illuminate brake light to warn vehicle in the behind.
	41,68%	AV does not maintain safe distance to structure.
	40,73%	Vehicle does not maintain safe distance from nearby objects.
(3) "Avoid flight in altitude below permitted limit."	**40,27%**	**Pilots must not climb/descent beyond designated altitude.**
	33,56%	The flight crew must never violate predetermined minimum/maximum altitude.
	30,58%	A/C must maintain minimum safe altitude limits.
	30,54%	Pilots must not stop maneuver before reaching designated altitude (except in emergency temination).
	27,47%	The flight crew must never violate the minimum distance to other aircraft.

portanto as três primeiras sugestões que envolvem controle de altitude (mas de sistemas variados) foram consideradas como relevantes.

4.5 Discussão

Em média, o primeiro passo do Pipeline atingiu o melhor resultado de classificação com 95,20% de acurácia e 95,08% de F1-Score, seguido pelo passo 2 com uma média de acurácia entre os três classificadores (Perda, Perigo e Restrição) de 88,51% e média de F1-Score de 86,27%. O passo 3, por sua vez, atingiu os resultados de classificação mais baixa entre todos os passos, com a média de acurácia de 83,44% e F1-Score de 73,16%.

O principal motivo para explicar estes resultados é a redução da quantidade de dados (tanto para treinamento quanto teste dos modelos) em cada passo sequencial do pipeline, refletido nos resultados pela queda dos valores de acurácia entre os modelos. O pipeline começa com todas as sentenças do conjunto de entrada no primeiro passo; este conjunto é então dividido em três subconjuntos para cada classe (Perda, Perigo e Restrição) para o segundo passo; por fim, cada um destes subconjuntos é dividido em conjuntos de sentenças corretas e incorretas, das quais apenas as sentenças incorretas são utilizadas como entrada no terceiro passo. Além de ter poucos dados de entrada, os modelos do passo 3 trabalham com a predição do maior número de rótulos entre os classificadores do pipeline (até 4 rótulos). Além disso, os detalhes presentes nas sentenças para cada tipo de erro podem ser pequenos e difíceis de capturar, o que reduz ainda mais a capacidade dos modelos de inferir o rótulo correto. Isto é refletido nas outras métricas além de acurácia como F1-Score, que obtiveram resultados ainda menores. As médias dos resultados do passo 3 se mantiveram próximas aos passos anteriores, pois o classificador de tipos de erros de Restrições se manteve consistente com todas as métricas acima de 95% de valor, o que balanceou os resultados das métricas de Perdas e Perigos abaixo de 80%.

Ao invés de utilizar um LLM baseado em *decoder* para sugerir uma paráfrase correta da sentença incorreta (o que poderia introduzir erros devido à geração de texto por máquina ou alucinações), foi optado neste trabalho pelo uso de um conjunto de sentenças corretas já existentes formado a partir do *dataset* criado na Seção 3.2, que teve a coluna de validação obtida por especialistas da análise STPA.

A vantagem desta abordagem é o modelo sempre sugerir apenas as sentenças corretas, sem espaço para introdução de erros. No entanto, esta abordagem é limitada pelo número de exemplos registrados no conjunto de sentenças corretas. Mesmo se o *dataset* for pequeno, ainda é possível sugerir correções para erros frequentes em domínios bem conhecidos. Em domínios menos conhecidos, existe a possibilidade de que não existam sugestões adequadas presentes no conjunto de sentenças corretas, o que resulta na sugestão de sentenças menos relevantes com taxas de similaridade mais baixas. Portanto, uma forma de mitigar este problema é incrementar a quantidade de sentenças corretas de domínios variados para expandir o leque de exemplos para sugestão. Como alternativa, foi fornecido ao usuário a porcentagem de similaridade entre a sentença incorreta e as sugestões corretas, para permitir que o usuário possa decidir se a sugestão é aceita, adaptada, ou recusada como referência para correção, com base na sua experiência.

4.6 Ameaças à validade dos experimentos

Existem duas principais ameaças à validade neste trabalho.

A primeira ameaça está presente na criação do *dataset*, onde foram incluídas novas informações sobre uma sentença com base no conhecimento do autor. Os dados adicionados são a coluna "*validation*", em que o autor determina se as sentenças são corretas ou incorretas, e a coluna "*error*", que define o tipo de erro encontrado nas sentenças incorretas. Esta ameaça é mitigada por ter a coluna "*validation*" revisada por especialistas em análise STPA, corrigindo rotulações erradas e garantindo que a sentença é realmente correta ou incorreta.

A segunda ameaça é o uso de modelos de classificação de aprendizado de máquina nos primeiros três passos do pipeline, e principalmente na classificação de sentenças corretas ou incorretas no segundo passo. Este problema é mitigado com o uso de validação cruzada dos dados (*K-Fold*), utilização de diversas métricas de avaliação (acurácia, precisão, sensibilidade e F1-Score), e fornecimento ao usuário das probabilidades de cada predição para suportar à decisão. No segundo passo do pipeline, para garantir que os dados usados durante o treinamento do modelo são confiáveis, os autores utilizaram o atributo "*validation*" previamente verificado por especialistas em análise STPA.

Capítulo 5

Conclusões

Algumas técnicas de análise de perigos, como a STPA, geram uma grande quantidade de texto e necessitam um esforço considerável dos analistas para verificar e corrigir os conteúdos gerados pela análise.

Este trabalho propôs um pipeline chamado BEDS (*BERT Error Detection for STPA*) para detecção de erros e sugerir correções para sentenças geradas durante o primeiro passo da análise STPA, e auxiliando o analista durante a análise.

Este pipeline é composto por quatro passos. O primeiro passo utiliza o modelo BERT para classificar sentenças não rotuladas em Perdas, Perigos em Nível de Sistema e Restrições em Nível de Sistema, para encontrar sentenças ambíguas ou erros de rotulação. O segundo passo utiliza três classificadores BERT, um para cada classe (Perda, Perigo e Restrição), para determinar se a sentença é correta ou incorreta. O terceiro passo também utiliza um classificador BERT para cada classe para identificar o tipo de erro presente na sentença incorreta classificada no passo anterior. O quarto e último passo utiliza um modelo de similaridade de sentenças *sentence-transformer* para sugerir sentenças corretas de um *dataset* validado.

Para treinar os modelos e testar o pipeline, foi criado um *dataset* de sentenças geradas durante a análise STPA por meio da extração de sentenças de fontes públicas. Os experimentos de classificação resultaram em 95.20% de acurácia para o primeiro passo, uma média de 88.51% de acurácia entre os classificadores de cada classe no segundo passo, e uma média de 83.44% de acurácia entre os classificadores do terceiro passo.

Pela análise dos resultados foi possível entender que sentenças de Perdas e Perigos encontram erros de classificação com mais frequência, pois existe uma diversidade maior

para expressar palavras e interpretar o contexto da análise. Restrições, por sua vez, possuem características de sentença mais definidas e são menos propícias a erros nos dois primeiros passos do pipeline.

Além disso, foi possível observar o impacto da variação da quantidade de dados para o pipeline. A cada passo, a quantidade de dados de entrada é reduzida, e isso é refletido pela acurácia decrescente ao longo do pipeline.

O potencial do pipeline proposto (BEDS) para auxiliar a análise STPA foi demonstrada pelos resultados obtidos nos testes realizados.

A classificação automática dos elementos do primeiro passo da STPA, como Perdas, Perigos e Restrições, pode aumentar a produtividade dos analistas de STPA e liberar mais tempo para os analistas focarem em tarefas mais complexas da análise (como o 2º passo da STPA).

Embora o uso de LLMs em sistemas críticos envolva diversas considerações de segurança e validação rigorosa, o uso de ferramentas está se provando uma alternativa viável para aumentar a produtividade e melhorar a qualidade das análises de perigo realizadas.

Os planos de trabalhos futuros incluem a expansão do *dataset*, para incluir dados de outras fontes e melhorar a qualidade de treinamento dos modelos; a criação de uma interface gráfica para interagir visualmente com o pipeline; a substituição dos modelos do pipeline por outros modelos além de BERT, para verificar as mudanças de desempenho; e a expansão do pipeline para os próximos passos da análise STPA.

Referências bibliográficas

ABDULKHALEQ, A.; WAGNER, S.; LEVESON, N. A Comprehensive Safety Engineering Approach for Software-Intensive Systems Based on STPA. en. **Procedia Engineering**, v. 128, p. 2–11, 2015. ISSN 18777058. DOI: 10.1016/j.proeng.2015.11.498. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S1877705815038588>.

ACAR CELIK, E. et al. Application of STPA for the Elicitation of Safety Requirements for a Machine Learning-Based Perception Component in Automotive. In: TRAPP, M.; SAGLIETTI, F.; SPISLÄNDER, M.; BITSCH, F. (Ed.). **Computer Safety, Reliability, and Security**. Cham: Springer International Publishing, 2022. v. 13414. P. 319–332. ISBN 9783031148347. DOI: 10.1007/978-3-031-14835-4_21. Disponível em: <https://link.springer.com/10.1007/978-3-031-14835-4_21>.

BAHDANAU, D.; CHO, K.; BENGIO, Y. **Neural Machine Translation by Jointly Learning to Align and Translate**. [S.l.: s.n.], 2016. arXiv: 1409.0473 [cs.CL]. Disponível em: <https://arxiv.org/abs/1409.0473>.

CARNIEL, A.; BEZERRA, J. D. M.; HIRATA, C. M. An Ontology-Based Approach to Aid STPA Analysis. **IEEE Access**, v. 11, p. 12677–12697, 2023. ISSN 2169-3536. DOI: 10.1109/ACCESS.2023.3242642. Disponível em: <https://ieeexplore.ieee.org/document/10037179/>.

CER, D. et al. SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. In: PROCEEDINGS of the 11th International Workshop on Semantic Evaluation (SemEval-2017). [S.l.]: Association for Computational Linguistics, 2017. DOI: 10.18653/v1/s17-2001. Disponível em: <http://dx.doi.org/10.18653/v1/S17-2001>.

CHARALAMPIDOU, S.; ZELESKIDIS, A.; DOKAS, I. M. Hazard analysis in the era of AI: Assessing the usefulness of ChatGPT4 in STPA hazard analysis. **Safety Science**, v. 178, p. 106608, 2024. ISSN 0925-7535. DOI: https://doi.org/10.1016/j.ssci.2024.106608. Disponível em: <https://www.sciencedirect.com/science/article/pii/S092575352400198X>.

DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018. DOI: 10.48550/ARXIV.1810.04805. Disponível em: <https://arxiv.org/abs/1810.04805>.

DIEMERT, S.; WEBER, J. H. **Can Large Language Models assist in Hazard Analysis?** [S.l.: s.n.], 2023. arXiv: 2303.15473 [cs.HC]. Disponível em: <https://arxiv.org/abs/2303.15473>.

FENG, X. et al. Application of natural language processing in HAZOP reports. **Process Safety and Environmental Protection**, v. 155, p. 41–48, 2021. ISSN 0957-5820. DOI: https://doi.org/10.1016/j.psep.2021.09.001. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957582021004675>.

FIELDS, J.; CHOVANEC, K.; MADIRAJU, P. A Survey of Text Classification With Transformers: How Wide? How Large? How Long? How Accurate? How Expensive? How Safe? **IEEE Access**, v. 12, p. 6518–6531, 2024. ISSN 2169-3536. DOI: 10.1109/ACCESS.2024.3349952. Disponível em: <https://ieeexplore.ieee.org/document/10380590/>.

FLEMING, C. H.; LEVESON, N. Integrating Systems Safety into Systems Engineering during Concept Development. en. **INCOSE International Symposium**, v. 25, n. 1, p. 989–1003, out. 2015. ISSN 2334-5837, 2334-5837. DOI: 10.1002/j.2334-5837.2015.00111.x. Disponível em: <https://incose.onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2015.00111.x>.

GHOSH, S.; ZABOLI, A.; HONG, J.; KWON, J. An Integrated Approach of Threat Analysis for Autonomous Vehicles Perception System. **IEEE Access**, v. 11, p. 14752–14777, 2023. ISSN 2169-3536. DOI: 10.1109/ACCESS.2023.3243906. Disponível em: <https://ieeexplore.ieee.org/document/10041909/>.

HARRIS, Z. S. Distributional Structure. en. ***WORD***, v. 10, n. 2-3, p. 146–162, ago. 1954. ISSN 0043-7956, 2373-5112. DOI: 10.1080/00437956.1954.11659520. Disponível em: <http://www.tandfonline.com/doi/full/10.1080/00437956.1954.11659520>.

HUGGING FACE. **Hugging Face Model Repository**. [S.l.: s.n.]. Disponível em: <https://huggingface.co/models?pipeline_tag=sentence-similarity>. Acesso em: 25 dez. 2024.

IKONOMAKIS, M.; KOTSIANTIS, S.; TAMPAKAS, V. et al. Text classification using machine learning techniques. **WSEAS transactions on computers**, v. 4, n. 8, p. 966–974, 2005.

JAMELL IVOR SAMUELS. One-Hot Encoding and Two-Hot Encoding: An Introduction. en, 2024. DOI: 10.13140/RG.2.2.21459.76327. Disponível em: <https://rgdoi.net/10.13140/RG.2.2.21459.76327>.

JOHRI, P. et al. Natural Language Processing: History, Evolution, Application, and Future Work. In: ABRAHAM, A.; CASTILLO, O.; VIRMANI, D. (Ed.). **Proceedings of 3rd International Conference on Computing Informatics and Networks**. Singapore: Springer Singapore, 2021. v. 167. P. 365–375. ISBN 9789811597114. DOI: 10.1007/978-981-15-9712-1_31. Disponível em: <http://link.springer.com/10.1007/978-981-15-9712-1_31>.

KOTSIANTIS, S. B.; ZAHARAKIS, I.; PINTELAS, P. et al. Supervised machine learning: A review of classification techniques. **Emerging artificial intelligence applications in computer engineering**, Amsterdam, v. 160, n. 1, p. 3–24, 2007.

LATIN American STAMP Workshop 2024. [S.l.: s.n.]. Disponível em: <https://www1.univap.br/la-stamp-workshop/program.html>. Acesso em: 25 dez. 2024.

LEVESON, N. A new accident model for engineering safer systems. en. **Safety Science**, v. 42, n. 4, p. 237–270, abr. 2004. ISSN 09257535. DOI: 10.1016/S0925-7535(03)00047-X. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0925753503000047X>.

LEVESON, N.; THOMAS, J. **STPA Handbook**. [S.l.: s.n.], 2018. Disponível em: <https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf>.

LEVESON, N. G. **Engineering a Safer World: Systems Thinking Applied to Safety**. [S.l.]: The MIT Press, jan. 2012. ISBN 9780262298247. DOI: 10.7551/mitpress/8179.001.0001. eprint: https://direct.mit.edu/book-pdf/2280500/book_9780262298247.pdf. Disponível em: <https://doi.org/10.7551/mitpress/8179.001.0001>.

MCCANN, B.; BRADBURY, J.; XIONG, C.; SOCHER, R. Learned in translation: Contextualized word vectors. **Advances in neural information processing systems**, v. 30, 2017.

MCGREGOR, J. Arcade game maker pedagogical product line: Concept of operations. **Version**, v. 2, p. 2005, 2005.

MELAMUD, O.; GOLDBERGER, J.; DAGAN, I. context2vec: Learning generic context embedding with bidirectional lstm. In: PROCEEDINGS of the 20th SIGNLL conference on computational natural language learning. [S.l.: s.n.], 2016. P. 51–61.

MIT Partnership for Systems Approaches to Safety and Security. [S.l.: s.n.]. Disponível em: <https://psas.scripts.mit.edu/home/>. Acesso em: 25 dez. 2024.

MOHER, D.; LIBERATI, A.; TETZLAFF, J.; ALTMAN, D. G. Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement. en. **International Journal of Surgery**, v. 8, n. 5, p. 336–341, 2010. ISSN 17439191. DOI: 10.1016/j.ijsu.2010.02.007. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S1743919110000403>.

PATIL, R.; BOIT, S.; GUDIVADA, V.; NANDIGAM, J. A Survey of Text Representation and Embedding Techniques in NLP. **IEEE Access**, v. 11, p. 36120–36146, 2023. ISSN 2169-3536. DOI: 10.1109/ACCESS.2023.3266377. Disponível em: <https://ieeexplore.ieee.org/document/10098736/>.

PETERS, M. E. et al. **Deep contextualized word representations**. [S.l.: s.n.], 2018. arXiv: 1802.05365 [cs.CL]. Disponível em: <https://arxiv.org/abs/1802.05365>.

PYTORCH. Disponível em: <https://pytorch.org/docs/stable/generated/torch.nn.functional.cosine_similarity.html>. Acesso em: 25 dez. 2024.

QI, Y.; ZHAO, X.; KHASTGIR, S.; HUANG, X. Safety Analysis in the Era of Large Language Models: A Case Study of STPA using ChatGPT, 2023. DOI: 10.48550/ARXIV.2304.01246. Disponível em: <https://arxiv.org/abs/2304.01246>.

RADFORD, A.; NARASIMHAN, K.; SALIMANS, T.; SUTSKEVER, I. et al. Improving language understanding by generative pre-training. OpenAI, 2018.

RAY, P. P. ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. en. **Internet of Things and Cyber-Physical**

Systems, v. 3, p. 121–154, 2023. ISSN 26673452. DOI: 10.1016/j.iotcps.2023.04.003. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S266734522300024X>.

REIMERS, N.; GUREVYCH, I. **Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks**. [S.l.: s.n.], 2019. arXiv: 1908.10084 [cs.CL]. Disponível em: <https://arxiv.org/abs/1908.10084>.

RISMANI, S. et al. From Plane Crashes to Algorithmic Harm: Applicability of Safety Engineering Frameworks for Responsible ML. en. In: PROCEEDINGS of the 2023 CHI Conference on Human Factors in Computing Systems. Hamburg Germany: ACM, abr. 2023. P. 1–18. ISBN 9781450394215. DOI: 10.1145/3544548.3581407. Disponível em: <https://dl.acm.org/doi/10.1145/3544548.3581407>.

SUN, C.; QIU, X.; XU, Y.; HUANG, X. How to Fine-Tune BERT for Text Classification? In: SUN, M. et al. (Ed.). **Chinese Computational Linguistics**. Cham: Springer International Publishing, 2019. v. 11856. P. 194–206. ISBN 9783030323806. DOI: 10.1007/978-3-030-32381-3_16. Disponível em: <http://link.springer.com/10.1007/978-3-030-32381-3_16>.

VARENOV, V.; GABDRAHMANOV, A. Security Requirements Classification into Groups Using NLP Transformers. In: 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW). [S.l.: s.n.], 2021. P. 444–450. DOI: 10.1109/REW53955.2021.9714713.

VASWANI, A. et al. Attention is all you need. **Advances in neural information processing systems**, v. 30, 2017.

WANG, A. et al. **GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding**. [S.l.: s.n.], 2019. arXiv: 1804.07461 [cs.CL]. Disponível em: <https://arxiv.org/abs/1804.07461>.