UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME SALES FERNANDES

AVALIAÇÃO DA CAPACIDADE COMPUTACIONAL E REPRESENTAÇÃO DE CONSCIÊNCIA DE UMA *CONSCIOUS TURING MACHINE*

FORTALEZA

2024

GUILHERME SALES FERNANDES

AVALIAÇÃO DA CAPACIDADE COMPUTACIONAL E REPRESENTAÇÃO DE CONSCIÊNCIA DE UMA *CONSCIOUS TURING MACHINE*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. João Fernando Lima Alcântara.

FORTALEZA

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F399a Fernandes, Guilherme Sales.
Avaliação da capacidade computacional e representação de consciência de uma Conscious Turing Machine / Guilherme Sales Fernandes. – 2025.
114 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Ciências, Curso de Computação, Fortaleza, 2025.
Orientação: Prof. Dr. João Fernando Lima Alcântara.

1. Máquina de Turing Consciente. 2. Inteligência artificial. 3. Arquiteturas conscientes. 4. Representação de consciência. 5. Explicabilidade. I. Título.

CDD 005

À minha incrível companheira, Ive Castro Barbosa, por acreditar em mim quando nem eu mesma acreditava. Seu apoio, amor e confiança foram a força que me sustentou nos momentos de dúvida e a luz que me guiou até aqui. Esta conquista também é sua.

AGRADECIMENTOS

Ao Prof. Dr. João Fernando Lima Alcântara, pela sabedoria, paciência e excelente orientação ao longo desta jornada, tanto no curso de graduação quanto no desenvolvimento deste trabalho. Nos momentos difíceis, ele me motivou e me mostrou que é possível conciliar nossos sonhos com a realidade, sendo um exemplo de professor que eu desejo seguir.

Aos professores participantes da banca examinadora Prof. Dr. Carlos Eduardo Fisch de Brito e Prof. Dr. Yuri Lenon Barbosa Nogueira, minha sincera gratidão por dedicarem seu tempo e conhecimento à avaliação deste trabalho e pelo compromisso em incentivar o desenvolvimento acadêmico com rigor e excelência.

Agradeço imensamente ao *Insight Data Science Lab*, laboratório do qual tive o privilégio de fazer parte durante quase toda minha graduação. Dentre os integrantes do *Insight*, preciso destacar aqueles que marcaram minha trajetória de forma especial: o Prof. Dr. José Antônio Fernandes de Macêdo, a Profa. Dra. Ticiana Linhares Coelho da Silva, a Profa. Dra. Lívia Almada Cruz e o Prof. Dr. Regis Pires Magalhães, cuja orientação e ensinamentos foram essenciais para meu aprendizado. Além deles, sou profundamente grato a Isabel Rabelo do Nascimento e Luiza Manoela Souza da Silva, cujo apoio e amizade tornaram essa jornada possível. A cada um de vocês e a todos os outros colegas de laboratório, meu muito obrigado por terem feito parte dessa caminhada.

À Miro Brodlova, desenvolvedora que realizou uma pesquisa na mesma área antes deste trabalho, mesmo sem me conhecer anteriormente, me recebeu com generosidade e disposição, contribuindo imensamente para o desenvolvimento deste trabalho. Sua ajuda foi essencial para a minha compreensão e aprofundamento no tema, e sou muito grato pela sua receptividade e apoio ao longo dessa jornada.

À minha companheira, Ive Castro Barbosa, por acreditar em mim nos momentos em que nem eu mesma acreditava. Seu amor, incentivo e paciência foram essenciais para que eu seguisse em frente, mesmo quando as dificuldades pareciam intransponíveis. Obrigada por ser minha base e minha motivação diária.

À minha terapeuta, Maria Domingas Câmara de Freitas, meu profundo agradecimento pelo seu imenso cuidado e profissionalismo. Seu apoio foi essencial para garantir minha saúde e bem-estar ao longo de todo este período desafiador.

À minha família, expresso minha gratidão pelo apoio ao longo desta jornada. Em especial, agradeço ao meu tio-avô, Marcos Alan da Silva Batista, pelo incentivo e ensinamentos,

aos meus avós, Regina Rosa Batista Sales e José Bonfim de Souza Sales; e minha mãe, Isabelle Kristine Batista Sales, pelo suporte e carinho nesse período. Muito obrigado a todos vocês.

Um agradecimento especial para todos os meus colegas da graduação, que foram parceiros e grandes aliados ao longo de toda essa jornada. Compartilhamos desafios, aprendizados e muitas madrugadas de estudo, e cada um de vocês tornou esse percurso mais leve e memorável. Obrigada pela companhia e pelo apoio durante todo o período.

Por fim, minha mais profunda gratidão e respeito a todos os trabalhadores e trabalhadoras que, com seu esforço diário, tornaram possível que eu concluísse esta graduação com qualidade e de forma gratuita. Aos professores da UFC, cuja dedicação à educação pública é um ato de resistência e transformação, e a toda equipe administrativa, que mantém essa universidade viva e acessível, mesmo diante de desafios constantes. Esta conquista não é apenas individual, mas fruto de uma coletividade de milhares de trabalhadores que seguem lutando, todos os dias, para que o conhecimento não seja um privilégio, mas um direito de nossa classe.

"O homem é apenas um individuo graças à sua memória intangível. Essa mesma memória não pode ser definida, mas apesar disso, ela define a humanidade."
(Masamune Shirow, 1989)

RESUMO

Com a rápida evolução de modelos generativos de texto e suas capacidades conversacionais, surgem muitas questões sobre a possibilidade de simular consciência nesses sistemas. Para abordar essa questão, é necessário enfrentar o desafio clássico da filosofia: compreender a natureza da consciência. Este trabalho explora essa problemática a partir da perspectiva da ciência da computação teórica, um campo dedicado a investigar os fundamentos da computação e da complexidade. Foi implementada em Python a arquitetura da *Conscious Turing Machine* (CTM), proposta por Manuel e Lenore Blum, com a qual foram conduzidos testes baseados em perguntas para avaliar a capacidade de compreensão e realização de inferências lógicas com o conjunto de dados *bAbI toy tasks* e a avaliação da capacidade de inferência textual co om conjunto de dados *RocStories*. Comparando o desempenho da CTM com modelos de linguagem individuais, mas que em conjunto formam os processadores da CTM. Os resultados mostram que, embora a CTM ofereça vantagens em termos de interpretabilidade e organização da informação, seu desempenho é semelhante ao de modelos menores e apresenta um custo computacional significativamente maior, além de apresentar divergências teóricas em relação às teorias clássicas da consciência. Concluímos que a CTM se destaca mais como uma metáfora computacional para o estudo da consciência do que como uma abordagem prática para Inteligência Artificial. Além disso, discutimos desafios de implementação e propomos direções para pesquisas futuras.

Palavras-chave: Arquiteturas Conscientes. *Conscious Turing Machine*. Explicabilidade. Representação de Consciência. Inteligência Artificial.

ABSTRACT

With the rapid evolution of generative text models and their conversational abilities, many questions arise regarding the possibility of simulating consciousness in such systems. Addressing this issue requires tackling the classic philosophical challenge of understanding the nature of consciousness. This study explores this problem from the perspective of theoretical computer science, a field dedicated to investigating the foundations of computation and complexity. The *Conscious Turing Machine* (CTM), proposed by Manuel and Lenore Blum, was implemented in Python, and experiments were conducted using question-based tests to assess logical inference capabilities with the *bAbI toy tasks* dataset and textual inference capabilities with the *RocStories* dataset. The CTM's performance was compared with that of individual language models, which collectively form the CTM processors. The results indicate that while the CTM provides advantages in interpretability and information organization, its performance is comparable to that of smaller models and comes with significantly higher computational costs. Furthermore, it presents theoretical divergences from classical theories of consciousness. We conclude that the CTM serves more as a computational metaphor for studying consciousness rather than as a practical approach to Artificial Intelligence. Additionally, we discuss implementation challenges and propose directions for future research.

Keywords: Conscious Architectures, *Conscious Turing Machine*, Explainability, Consciousness Representation, Artificial Intelligence.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CTM	Concious Turing Machine
GWT	Global Workspace Theory
IA	Inteligência Artificial
LTM	Long Term Memory
PLN	Processamento de Linguagem Natural
STM	Short Term Memory

SUMÁRIO

1 INTRODUÇÃO

Desde a origem da computação como ciência, há a indagação sobre as possíveis capacidades de uma Inteligência Artificial (IA). Um dos artigos fundadores é de Alan Turing, no trabalho Computing Machinery and Intelligence, Turing (1950) onde propõe as bases dessa nova área, construindo uma longa reflexão fundamentada na pergunta "Podem as máquinas pensar?". Um dos pontos principais desse trabalho está na sua abordagem para responder à questão central. Sua linha de pensamento é construída partindo da ideia da grande dificuldade na definição do que significa "pensar"e o que é uma "máquina"; então, é proposta uma prova por analogia, em que, ao "substituir a questão por outra que esteja intimamente relacionada a ela e seja expressa em palavras relativamente inequívocas", desenvolve-se uma nova abordagem para pensar sobre essa questão. Para isso, é necessário, primeiro, encontrar uma ideia simples e inequívoca para substituir a palavra "pensar"; segundo, deve-se explicar exatamente quais "máquinas"são consideradas e, por fim, munido dessas ferramentas, formular uma nova questão, relacionada à primeira, que possa responder de forma afirmativa.

Como existe uma complicação conceitual na solução proposta (Turing, 1950, p. 447), é assim criado um experimento mental chamado de "O Jogo da Imitação". A intenção desse experimento é investigar a capacidade de uma máquina exibir comportamento inteligente indistinguível do humano. No teste, um interrogador humano conversa, por meio de um terminal, com dois participantes ocultos: um humano e uma máquina. O objetivo da máquina é enganar o interrogador, fazendo-o acreditar que ela é humana, enquanto o humano tenta demonstrar sua própria identidade. Dessa forma, a questão da avaliação da Inteligência Artificial é abordada por um novo viés, nesse caso, um viés relacional, em que o foco é deslocado do conceito das entidades para o vínculo entre entrevistador e entrevistado.

Esse experimento traz consigo uma postura fenomenológica ao se debruçar sobre como as interações com estas entidades ocultas aparecem na e para a consciência do tempo do entrevistador. Isso significa que, neste experimento, apesar de ter como objetivo identificar qual dos entrevistados é humano e qual é a máquina, não é relevante determinar se a interação entre as partes corresponde ou não a objetos do mundo externo à mente do entrevistador. O foco está no modo como ele experiencia e constrói o conhecimento sobre as entidades, e não nas entidades que existem de forma independente. Assim, concentrando-se exclusivamente na experiência em si, pois é essa experiência que constitui sua realidade e que trará a capacidade de julgar qual das entidades é humana, independente de ser ou não de fato.

Com o experimento proposto por Turing, fica evidente esse problema fundamental que é a definição de inteligência por extrapolação, a definição da própria consciência, já que é um conceito ainda mais abstrato e multifacetado do que inteligência. Para circundar a questão da compreensão intrínseca ou genuína desse conceito, a alternativa encontrada está em verificar a capacidade de gerar interações convincentes para um entrevistador humano. Mais do que estudar se sistemas de IA podem ser conscientes, o experimento é uma reflexão sobre como a humanidade relaciona-se com essas entidades e projeta nelas características humanas. Dessa forma, o estudo de interações com esses sistemas acaba sendo, em grande medida, um estudo sobre a natureza humana: como percebidas e interpretadas essas relações, quais expectativas são criadas e como as próprias noções de inteligência, comunicação e identidade são moldadas e desafiadas por essas novas formas de interação.

Partindo desse preceito de que uma entidade é consciente apenas pela validação de uma outra - já que sua definição é de natureza subjetiva - e que esse entendimento é necessariamente intermediado por uma relação, este trabalho adota uma postura antropocêntrica, ao descartar a diferenciação de "consciência real"do que "parece ser consciência"ou "consciência virtual"a partir da nossa perspectiva humana.

Neste trabalho, será utilizada uma formalização da arquitetura de *Conscious Turing Machine* (CTM), proposta por Lenore e Manuel Blum, Blum and Blum (2022). Este modelo se apresenta como um “modelo único, abstrato e independente do substrato de consciência” (Blum and Blum, 2022, p. 1) e busca abordar a consciência sob uma perspectiva da ciência da computação teórica, utilizando-se de conceitos como a *Global Workspace Theory* (GWT), Baars (2005), desenvolvida por Bernard Baars. A Concious Turing Machine (CTM) é uma representação computacional que formaliza matematicamente e combina processos cognitivos de memória de curto e longo prazo para realizar tarefas conscientes e inconscientes, fornecendo uma plataforma robusta para investigar fenômenos associados à consciência.

A importância de investigar a CTM como uma possibilidade para estudar a consciência está enraizada em seu potencial para unificar conceitos teóricos de ciência da computação e de atividades cognitivas. Este modelo contribui para o avanço de abordagens arquiteturais na IA, proporcionando uma abordagem para compreender a interação entre processamento consciente e inconsciente.

Compreender as limitações e os pontos fortes de arquiteturas inspiradas em processos cognitivos humanos oferece uma oportunidade para desenvolver sistemas de IA mais intuitivos,

explicáveis e eficientes. Este trabalho é relevante não apenas como uma prova de conceito, mas também como um estudo empírico que avalia um aspecto da capacidade prática do modelo.

1.1 Objetivos

Este projeto de pesquisa tem como objetivo geral oferecer uma implementação formal de uma CTM e, a partir disso, medir a eficácia em tarefas estruturadas de lógica e inferência textual, além de fundamentar uma discussão sobre problemas e possíveis melhorias para esta arquitetura. Para abranger esses pontos, as avaliações terão foco em dois cenários distintos:

1. A avaliação de tarefas lógicas utilizando o conjunto de dados bAbI Toy Tasks de Weston et al. (2015)
2. A avaliação da capacidade de inferência em textos narrativos do RocStories de Mostafazadeh et al. (2016)

Através desses experimentos, apresentam-se os objetivos específicos deste trabalho, sendo eles:

1. Fornecer uma implementação da arquitetura da CTM em Python, considerando as especificações teóricas descritas por Manuel e Lenore Blum, já que até a data deste trabalho é um projeto de caráter apenas teórico.
2. Avaliar o desempenho da arquitetura implementada em tarefas de lógica e inferência textual.
3. Apontar problemas e possíveis melhorias desta arquitetura.

1.2 Visão Geral

Este trabalho tem como objetivo explorar e formalizar a implementação da CTM. Para isso, este trabalho aborda no **Capítulo 2**, apresentamos a **Fundamentação Teórica**, onde são discutidos os conceitos necessários relacionados à consciência, incluindo o *Problema Difícil da Consciência*, a GWT e a definição da *Conscious Turing Machine* (CTM). Além disso, são detalhadas suas principais estruturas e dinâmicas.

O **Capítulo 4** é dedicado à **Metodologia**, na qual descrevemos a principal contribuição deste trabalho: uma implementação formal da CTM em Python. São discutidos os aspectos técnicos necessários para a implementação, os desafios enfrentados, as adaptações realizadas para manter a fidelidade ao modelo teórico original e as soluções empregadas para viabilizar

a implementação. Ainda neste capítulo, detalhamos o procedimento adotado para a revisão bibliográfica. Ao final desse capítulo, são apresentados os **experimentos** conduzidos, incluindo os resultados da revisão bibliográfica e a análise dos experimentos realizados sobre a CTM utilizando os conjuntos de dados *bAbI Toy Tasks* (para perguntas lógicas) e *RocStories* (para inferência textual). Esses experimentos foram conduzidos com o intuito de avaliar a capacidade do modelo implementado em simular aspectos de raciocínio e inferência, sinais considerados importantes na representação da consciência.

No **Capítulo 5**, apresenta os **resultados** dos experimentos realizados. Para avaliar a performance dos modelos, a principal métrica utilizada foi a acurácia, permitindo uma comparação quantitativa da efetividade da CTM em diferentes cenários experimentais. Além disso, investigamos a capacidade de explicabilidade do modelo, analisando como suas estruturas internas justificam as decisões tomadas ao longo do processamento das tarefas. Para ilustrar os resultados, apresentamos gráficos e tabelas que demonstram a evolução do desempenho.

Por fim, no **Capítulo 5**, trazemos a **Conclusão** e os **Trabalhos Futuros**, onde discutimos as implicações do trabalho realizado, analisamos suas limitações e sugerimos direções para pesquisas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

O objetivo deste capítulo estabelece-se em apresentar a fundamentação teórica que embasa o trabalho apresentado, além de fornecer o contexto necessário para a análise e discussão das arquiteturas implementadas e resultados obtidos. Com isso, nas próximas sessões serão apresentados os conceitos de Consciência, o Problema Difícil da Consciência; Arquiteturas Conscientes; *Global Workspace Theory*; *Conscious Turing Machine* e a linguagem *Brainish*. Esses conceitos serão necessários para a execução das experimentações realizadas e compreensão teórica.

2.1 Consciência

A reflexão sobre a consciência é tema de estudo desde o nascimento da filosofia grega clássica, apesar de não ter sido desenvolvida com o termo exato, "consciência". Platão, em seu trabalho Fédon (Platão, 2016), dialoga sobre a imortalidade da alma, a natureza do corpo e da alma, e a busca pelo conhecimento verdadeiro. Nesse diálogo, a relação semântica entre alma e consciência é profunda, o que faz dele também uma reflexão sobre a natureza da consciência.

É evidente tal relação ao observar os trabalhos de Aristóteles, que foi seu discípulo na Academia de Atenas. No trabalho *De Anima* (Aristóteles, 2023), Aristóteles desenvolve a análise sistemática da alma —*psique* — e suas funções. Este trabalho tem relevância central no estudo da consciência porque investiga a relação entre a alma e as capacidades cognitivas, como percepção, pensamento e razão, que estão diretamente ligadas ao conceito de consciência. Na análise proposta, é necessário destacar as seguintes pontuações:

1. A alma como princípio vital, desse modo, rejeitando a visão platônica de uma separação radical entre alma e corpo, argumentando que a alma e o corpo estão intrinsecamente conectados. A consciência, nesse contexto, emerge da interação entre ambos.
2. Sobre as Faculdades da Alma, onde é dividida a alma em diferentes faculdades, cada uma com funções específicas, que ajudam a entender a consciência: Vegetativa, sendo responsável pelo crescimento e nutrição, presente em plantas, animais e humanos. Sensitiva, relacionada à percepção sensorial e movimento, exclusiva de animais e humanos. Racional, sendo a faculdade mais elevada, responsável pelo pensamento, reflexão e raciocínio, exclusiva dos humanos. A consciência é mais amplamente vinculada à faculdade racional, mas dependente das faculdades inferiores para processar informações sensoriais.

3. Sobre a percepção e a consciência, Aristóteles distingue a percepção — *aisthesis* — da consciência, conceituando a percepção como uma forma primitiva da verdadeira consciência, pois desempenha o papel de apenas reagir aos estímulos externos. A consciência é então de natureza mais elevada, pois tem a função de processar a percepção pela razão, gerando reflexões a partir desta.
4. O intelecto passivo e o intelecto ativo. O intelecto passivo é responsável por receber e armazenar informações sensoriais, funcionando como um repositório das percepções captadas pelos sentidos. Por outro lado, o intelecto ativo é a capacidade de abstração e pensamento puro, permitindo a compreensão das essências e a reflexão sobre conceitos universais. É nesse nível que a consciência atinge seu grau mais elevado, evidenciando a interação entre os aspectos sensoriais e racionais da alma.
5. A autoconsciência, que sugere que a alma racional é capaz de refletir sobre si mesma, indicando uma forma inicial de autoconsciência. Essa capacidade de pensar sobre o próprio pensamento é central na obra tratada.

Seguindo adiante na cronologia do trabalho, tem-se René Descartes, que publica em 1641 As Meditações sobre a Filosofia Primeira — *Meditationes de Prima Philosophia* — (Descartes, 2004), sendo uma das obras mais influentes para a filosofia ocidental. Esse trabalho busca estabelecer fundamentos seguros para o conhecimento, questionando as bases da realidade, da existência e da mente humana. No contexto da consciência, Descartes apresenta reflexões profundas sobre a natureza do pensamento, da percepção e da relação entre mente e corpo. Desse trabalho, será usada em específico a meditação de número dois, que pontua o argumento mais famoso desta obra, "Penso, logo existo"— *Cogito, ergo sum*. A filosofia cartesiana tem como uma de suas bases aplicar a dúvida sistemática, descartando tudo o que pode ser falso ou incerto; dentre essas possibilidades estão as percepções sensoriais e a existência do corpo, e por isso, poderia também duvidar do que está pensando. Um problema lógico estabelece-se ao duvidar recursivamente de tudo, pois é possível também duvidar da própria dúvida originária, tornando o processo de construção de conhecimento infundado. Essa incongruência poderia tornar sua teoria contraditória, reduzindo-a ao absurdo, ou infinitamente recursiva, sendo incapaz de chegar a uma conclusão por sempre duvidar de qualquer afirmação possível. Devido a isso, é fundamentada uma base recursiva para esse pensamento que é o de não poder duvidar de que está pensando. O ato de duvidar, refletir ou questionar é, em si, uma prova da existência do eu pensante. Esse "eu"é a consciência, a essência da mente e do ser. Sendo assim, a consciência é o fundamento do ser, o

que torna o pensamento (e, portanto, a consciência) a base de toda existência e conhecimento.

Os pontos apresentados fundamentam os estudos ao definir esses dois principais pontos: A mente é a sede da consciência; autoconsciência é a essência do ser. A partir disso, a área de computação cognitiva e filosofia da mente traçam novos caminhos para construírem uma definição objetiva de consciência, e a partir desta definição, simular um comportamento semelhante em um ambiente controlado.

Essas simulações são direcionadas através da implementação de arquiteturas conscientes; essas, por sua vez, são implementações que fundamentam-se na filosofia abordada nesta seção, representando o conhecimento formado em um sistema lógico. As próximas seções mergulharão nos detalhes da arquitetura da CTM, delineando as abordagens, componentes fundamentais e as interações que a fundamentam.

2.2 O Problema Difícil da Consciência

O Problema Difícil da Consciência, introduzido por Chalmers (2007b), refere-se à questão fundamental de como e por que estados físicos, como atividades cerebrais, geram experiências subjetivas, conhecidas como *qualia*. Enquanto problemas "fáceis" da consciência tratam de funções cognitivas que podem ser explicadas por mecanismos computacionais ou neurológicos, como percepção, memória e atenção, o problema difícil foca na natureza da experiência consciente: *Por que há algo que é "como" ser consciente?*

Chalmers argumenta que mesmo mapeando todos os correlatos neurais da consciência, ainda restará uma lacuna explicativa entre o funcionamento físico do cérebro e a experiência subjetiva. Essa lacuna é o que diferencia o problema difícil dos problemas funcionais. Em outras palavras, o desafio está em explicar como processos físicos, como disparos de neurônios, resultam em experiências internas como o "vermelho" de uma maçã ou a "doçura" do açúcar.

2.3 *Global Workspace Theory* (GWT)

A Global Workspace Theory (Teoria do Espaço Global), proposta por Baars (2005), é amplamente reconhecida como uma das abordagens mais robustas para compreender o funcionamento da consciência. Baseada em princípios da arquitetura cognitiva e em analogias com o cérebro humano, a GWT descreve a consciência como um sistema que integra e dissemina informações, permitindo a interação eficiente entre diferentes módulos cognitivos. Sua força

reside em explicar como o conteúdo consciente emerge e se dissemina pelo cérebro, facilitando a coordenação de processos cognitivos inconscientes.

2.3.1 Metáfora do Teatro da Consciência

A GWT utiliza a poderosa metáfora do **teatro mental** para ilustrar como a consciência funciona:

- O **palco** representa a *memória de trabalho*, onde os conteúdos conscientes são exibidos de forma temporária.
- O **holofote de atenção** ilumina os conteúdos mais relevantes no palco, permitindo que informações críticas sejam destacadas para processamento consciente.
- O **público** é composto por *processadores inconscientes*, que observam as informações exibidas no palco e podem agir com base nelas.

Essa metáfora do teatro destaca como apenas uma pequena fração das informações processadas pelo cérebro chega ao estado consciente, enquanto a maioria permanece nos bastidores, atuando de maneira inconsciente. A partir dessa estrutura, a GWT fornece um modelo funcional para entender como conteúdos conscientes são gerados e amplamente compartilhados entre diferentes subsistemas cerebrais.

2.3.2 Limitações e Críticas

Embora a GWT seja amplamente aceita, ela enfrenta algumas críticas importantes que apontam para desafios em sua validação empírica e abrangência explicativa:

2.3.2.1 Falta de Precisão Neuronal

Apesar de existirem evidências de que a consciência está associada à ativação ampla de regiões como o córtex pré-frontal e parietal Pal et al. (2018), a Global Workspace Theory (GWT) carece de detalhes sobre os mecanismos exatos pelos quais o espaço global opera no nível neuronal.

2.3.2.2 Subjetividade da Experiência

A GWT explica como as informações são disseminadas e integradas, mas não aborda diretamente o “problema difícil” da consciência, que envolve explicar como surgem as qualidades

subjetivas (ou *qualia*) da experiência.

2.3.2.3 Ambiguidade do Diretor Central

A metáfora do teatro sugere a presença de um diretor que controla o que se torna consciente. Contudo, em implementações modernas da GWT, como na *Conscious Turing Machine* (CTM), Blum and Blum (2022), discute-se se o papel do diretor é necessário ou se a consciência pode emergir de processos distribuídos sem controle centralizado.

2.4 *Conscious Turing Machine* (CTM)

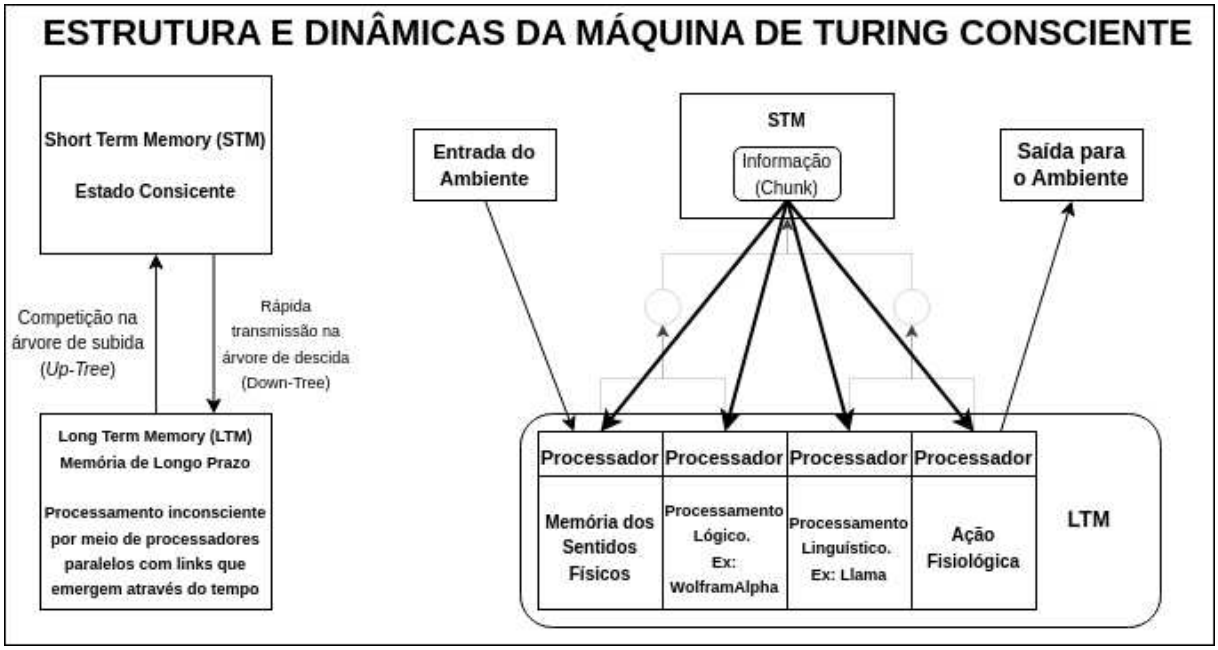

Figura 1 – Arquitetura original da CTM adaptada do trabalho original de Blum and Blum (2022)

A *Conscious Turing Machine* (Máquina de Turing Consciente) proposta por Blum and Blum (2022), é um modelo teórico que busca formalizar a consciência sob uma perspectiva computacional. A CTM recebe o sintagma nominal *"Turing Machine"* (TM) — Máquina de Turing — por carregar uma forte similaridade com a tradição da computação teórica. A escolha desse termo representa que a CTM compartilha com a TM a característica de ser um sistema computacional formalmente definido por um modelo matemático "simples", porém poderoso, sendo assim uma extensão do conceito de TM, aplicada a um domínio diferente, o estudo da consciência. A CTM diferencia-se de uma TM tradicional porque sua arquitetura é também inspirada pela *Global Workspace Theory* (GWT), em que seu funcionamento e comportamento

consciente emergem não do poder computacional bruto ou da transformação de entrada em saída, mas da organização estrutural da GWT. A CTM é construída para simular processos conscientes e inconscientes em uma arquitetura computacional orientada pela definição de Baars. Como mostrado na Imagem 1, a formalização da CTM é dada como uma **7-tupla** $\langle \text{STM}, \text{LTM}, \text{Down-Tree}, \text{Up-Tree}, \text{Links}, \text{Input}, \text{Output} \rangle$, em que cada componente desempenha um papel específico em um intervalo de tempo discreto interno do sistema chamado de *tick*:

- ***Short Term Memory*** (STM): Memória responsável por armazenar os conteúdos conscientes que emergem da *LTM* por meio da *Up-Tree*
- ***Long Term Memory*** (LTM): Memória responsável por controlar os processadores inconscientes que irão gerar *Chunks* (informações) para disputar o estado consciente na *Up-Tree*
- ***Down-Tree***: A *Down-Tree* é a estrutura de dados que permite que a informação armazenada pela STM (consciente) seja transmitida para todos os processadores da LTM.
- ***Up-Tree***: A *Up-Tree* é uma árvore binária que recebe como entrada os *Chunks* produzidos pelos processadores da LTM. Esses *chunks* disputam a posição na STM subindo a*Up-Tree*.
- ***Links***: Os *Links* são relacionamentos bidirecionais entre os processadores da LTM, eles são formados quando há interações entre as informações gerads por eles na STM. Eles servem para os processadores poderem comunicar-se sem precisar enviar informações para a STM.
- ***Input*** O *Input Map* ou apenas *Input*, é a camada responsável por transmitir informações ambiente para dentro da CTM.
- ***Output*** O *Output Map* ou apenas *Output*, é a camada responsável por transmitir informações conscientes da CTM em respostas ao ambiente.

Nas próximas seções, serão abordados os seguintes tópicos: As estruturas e dinâmicas da CTM; e a linguagem *Brainish*, que é a linguagem interna original da CTM;

2.4.1 Estruturas Principais da CTM

As estruturas principais da CTM formam a base para sua modelagem teórica da consciência, combinando conceitos inspirados pela GWT. Nesta seção, serão explorados os elementos fundamentais que compõem a CTM, incluindo a Memória de Curto Prazo (*Short-Term Memory* - STM), responsável por armazenar e transmitir o conteúdo consciente; a Memória de Longo Prazo (*Long-Term Memory* - LTM), que coordena o processamento distribuído; as

Árvores Direcionais (*Up-Tree* e *Down-Tree*), que organizam competições e transmissões de informações; e os *Links*, que facilitam a troca de *chunks* entre os processadores.

2.4.1.1 Chunk

Os *chunks* são a menor unidade de informação manipulada pela CTM e desempenham um papel central na dinâmica do modelo. Cada *chunk* encapsula uma quantidade limitada de informação que pode representar percepções, pensamentos, respostas ou comandos.

Um *chunk* é formalmente definido como uma 6-upla:

$$\langle \text{Address}, \text{Time}, \text{Gist}, \text{Weight}, \text{Intensity}, \text{Mood} \rangle$$

- Endereço (Address): Identifica o processador da LTM que gerou o *chunk*.
- Tempo (Time): Registra o momento em que o *chunk* foi produzido.
- Essência (Gist): Representa o conteúdo principal do *chunk*, como uma ideia, sensação ou resposta.
- Peso (Weight): Um valor numérico que indica a relevância ou importância atribuída ao *chunk* pelo processador que o gerou.
- Intensidade (Intensity): O valor absoluto do peso, refletindo a urgência ou prioridade do *chunk*.
- Humor (Mood): O sinal do peso, indicando se o *chunk* é percebido como positivo (otimista) ou negativo (pessimista).

2.4.1.2 Short Term Memory (STM)

Short Term Memory (Memória de Curto Prazo) da CTM, é a estrutura que, a cada tick $t = 0, 1, 2, \ldots$, armazena exatamente um *chunk* (uma unidade de informação). O Short Term Memory (STM) funciona como o "palco" da CTM, onde o *chunk* vencedor da competição na *Up-Tree* torna-se consciente. Esse *chunk* é então transmitido para todos os processadores da *Long Term Memory* (LTM) por meio da *Down-Tree*, garantindo que o conteúdo consciente seja amplamente difundido. O STM é central para a definição de *conscious awareness* (percepção da consciência) na CTM, pois representa o conteúdo consciente em cada instante.

2.4.1.3 Long Term Memory (LTM)

Long Term Memory (Memoria de Longo Prazo) é a estrutura composta por uma coleção de processadores que operam em paralelo, cada um com sua própria memória. Esses processadores funcionam de forma inconsciente, produzindo *chunks* que competem para se tornar conscientes. A Long Term Memory (LTM) é responsável pelo processamento pesado de informações, gerando ideias, respostas e sensações que podem ser transmitidas ao STM. Com o tempo, processadores da LTM podem formar *links* diretos, permitindo comunicação eficiente e aprendizado associativo.

2.4.1.4 Down-Tree

A *Down-Tree* (Árvore de Descida) é uma estrutura de transmissão que parte do STM e se ramifica para alcançar todos os processadores da LTM. Sua função é garantir que o *chunk* vencedor da *Up-Tree* seja simultaneamente difundido para todos os processadores, sincronizando o conteúdo consciente.

2.4.1.5 Up-Tree

A *Up-Tree* (Árvore de Subida) é uma estrutura de competição que organiza os *chunks* produzidos pelos processadores da LTM. Cada processador insere seu *chunk* na base da *Up-Tree*, e uma competição ocorre em cada nível da árvore até que um único *chunk* alcance a raiz *STM*. Essa competição é baseada em critérios como o peso e a intensidade do *chunk*, decidindo qual será transmitido como conteúdo consciente.

2.4.1.6 Links

Os *links* são conexões dinâmicas que se formam entre processadores da LTM ao longo do tempo. Eles permitem comunicação direta entre processadores, reduzindo a necessidade de passar pelo STM. Links fortalecem-se com interações repetidas, facilitando aprendizado e adaptação.

Essas estruturas relacionam-se de maneiras diversas e geram dinâmicas diferentes para promover o estado de *concious-awareness*.

2.5 Dinâmicas da CTM

As dinâmicas da CTM representam os processos internos que dão vida à sua arquitetura, permitindo que o sistema simule aspectos centrais da consciência. Nesta Seção, serão exploradas as principais dinâmicas que estruturam essa arquitetura.

2.5.1 Competição e Seleção na Up-Tree

Os *chunks* produzidos pelos processadores da LTM competem na *Up-Tree* para determinar qual deles será transmitido ao STM. Esse processo ocorre de maneira hierárquica, com comparações em cada nível da árvore, até que o *chunk*vencedor alcance a raiz. Há dois tipos de competição: Determinística e Probabilística.

2.5.2 Função de Competição Determinística

Na versão determinística, a função de competição f é aplicada para comparar dois *chunks* em um nó da Up-Tree. O *chunk*com o maior valor de f avança para o próximo nível da árvore.

Sejam dois *chunks* C_1 e C_2, com os seguintes componentes:

- $C_1 = \langle \text{address}_1, \text{time}_1, \text{gist}_1, \text{weight}_1, \text{intensity}_1, \text{mood}_1 \rangle$
- $C_2 = \langle \text{address}_2, \text{time}_2, \text{gist}_2, \text{weight}_2, \text{intensity}_2, \text{mood}_2 \rangle$

A função de competição f pode ser definida, por exemplo, como:

$$f(C) = \text{intensity} + \frac{1}{2} \cdot \text{mood}.$$

A seleção do *chunk*vencedor é então determinada por:

$$\text{Se } f(C_1) > f(C_2), \text{ então } C_1 \text{ avança.}$$

$$\text{Se } f(C_1) = f(C_2), \text{ o } \textit{chunk} \text{ com menor endereço avança.}$$

Essa abordagem garante que o processo de seleção seja uniforme e determinado unicamente pelos valores atribuídos aos *chunks*.

2.5.3 Função de Competição Probabilística

Na versão probabilística, a função de competição é modificada para incluir um elemento de aleatoriedade. Nesse caso, o avanço de um *chunk* não é garantido apenas pelo valor

de f, mas também por uma distribuição de probabilidade associada a cada *chunk*.

A probabilidade de um *chunk* C_i avançar é proporcional ao seu valor de $f(C_i)$. Assim, para dois *chunks* C_1 e C_2, a probabilidade de C_1 ser selecionado é dada por:

$$P(C_1 \text{ avança}) = \frac{f(C_1)}{f(C_1)+f(C_2)},$$

e a probabilidade de C_2 ser selecionado é:

$$P(C_2 \text{ avança}) = \frac{f(C_2)}{f(C_1)+f(C_2)}.$$

Esse modelo permite que *chunks* com valores ligeiramente inferiores ainda tenham uma chance de avançar, promovendo uma competição mais flexível e reduzindo a rigidez da seleção determinística.

2.5.4 Exemplo de Aplicação

Suponha que temos dois *chunks*, C_1 com $f(C_1) = 10$ e C_2 com $f(C_2) = 6$. Na função determinística, C_1 sempre avançará, pois $f(C_1) > f(C_2)$. No entanto, na função probabilística, as probabilidades de avanço são:

$$P(C_1 \text{ avança}) = \frac{10}{10+6} = \frac{10}{16} = 0.625,$$
$$P(C_2 \text{ avança}) = \frac{6}{10+6} = \frac{6}{16} = 0.375.$$

Nesse caso, C_2 ainda tem 37,5% de chance de avançar, mesmo com um valor de f inferior.

2.5.5 Importância da Função de Competição

A função de competição, tanto na versão determinística quanto na probabilística, determina quais informações se tornam conscientes ao alcançar o STM e influencia diretamente o *stream of consciousness* gerado pela máquina. A escolha entre um modelo determinístico ou probabilístico depende do nível de flexibilidade e adaptabilidade desejado para o sistema.

2.5.6 Transmissão na Down-Tree

Após a seleção na *Up-Tree*, o *chunk* vencedor é transmitido para todos os processadores da LTM através da *Down-Tre*e. Essa transmissão garante que todos os processadores compartilhem a mesma informação consciente, promovendo integração e coordenação.

2.5.7 Formação e Fortalecimento de Links

Os *links* entre processadores da LTM permitem comunicação eficiente e direta. Quando dois processadores interagem frequentemente de maneira útil, os *links* se fortalecem, facilitando a troca de informações e reduzindo a carga do STM, fazendo um contato direto.
O desenvolvimento da CTM seguiu, com os artefatos disponíveis no estado da arte, as diretrizes teóricas de Blum & Blum (2021), garantindo uma arquitetura modular alinhada à *Global Workspace Theory*. A implementação das estruturas *STM*, *LTM*, *Up-Tree* e *Down-Tree* permitiu um fluxo de informação eficiente, enquanto mecanismos auxiliares, como o *DiscreteClock* e o *Output Map*, asseguraram a coordenação e o registro dos processos cognitivos. Essa metodologia fornece uma base robusta para a modelagem computacional da consciência.

2.6 A Linguagem Brainish

O *Brainish* de Liang (2022) é uma linguagem interna multimodal. Idealmente, ela deveria ser utilizada pela CTM, desempenhando um papel crucial na coordenação e comunicação entre os processadores da CTM, facilitando funções cognitivas complexas, como processamento multimodal, modelo mental do mundo, visão interna, sensação tátil, e até mesmo sonhos. Essa linguagem foi projetada para integrar diferentes modalidades de dados, incluindo texto, imagens, áudio e sensações, em representações coordenadas e compreensíveis pelo sistema, agregando essas informações em um mesmo espaço latente.

2.6.1 Definição e Objetivo do Brainish

O objetivo principal do *Brainish* é fornecer uma base unificada para representar e processar informações oriundas de diferentes modalidades sensoriais. Ele é composto por três componentes principais:

- *Encoders* Unimodais: Segmentam e representam dados de uma única modalidade (ex.: texto ou imagens).
- Espaço de Representação Coordenado: Relaciona e compõe características unimodais para derivar significados holísticos a partir de entradas multimodais.
- *Decoders*: Mapeiam representações multimodais para dados preditivos (fusão) ou dados brutos (tradução ou geração).

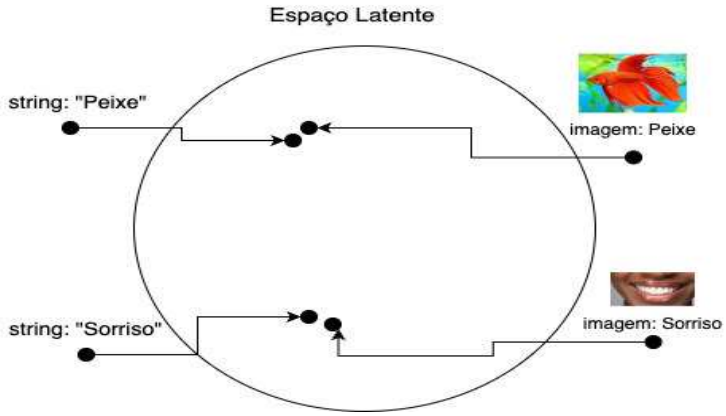

Figura 2 – Representação de Informações no *Brainish*

2.6.2 Estado Atual do Brainish

Com a última atualização de Liang (2022, p. 20), o *Brainish* é um *framework* em desenvolvimento. Devido à falta de exemplos, indisponibilidade de acesso a qualquer implementação e ao escopo trabalhado neste projeto, a linguagem interna da CTM será alterada. Maiores explicações estão no capítulo de metodologia.

2.6.3 Importância do Brainish na CTM

De acordo com Blum and Blum (2022, p. 4), o *Brainish* é fundamental para realizar funções cognitivas complexas na CTM, permitindo que os processadores se comuniquem de forma eficiente e coordenada. Ele facilita a integração de diferentes tipos de informação ao representá-los em um mesmo espaço latente, como é exemplificado na Imagem 2

2.7 *Large Language Models* (LLMs)

Os Modelos de Linguagem de Grande Escala, conhecidos como *Large Language Models* (LLMs), representam uma classe de modelos de inteligência artificial baseados em aprendizado profundo, treinados em grandes quantidades de dados textuais para realizar diversas tarefas de Processamento de Linguagem Natural (PLN). Esses modelos são construídos a partir de arquiteturas de redes neurais profundas, sendo a mais comum a arquitetura *Transformer*, introduzida por Vaswani et al. (2023).

2.7.1 Arquitetura e Funcionamento

A base dos LLMs é a arquitetura Transformer, que se destaca pelo uso do mecanismo de autoatenção (*self-attention*) e normalização por camadas (*layer normalization*) para processar sequências textuais de maneira eficiente e paralelizada. Diferentemente de abordagens anteriores, como Redes Neurais Recorrentes (RNNs) e Redes Neurais Convolucionais (CNNs), os Transformers eliminam a dependência sequencial direta, permitindo treinamento em larga escala com maior eficiência computacional.

A arquitetura Transformer é composta por múltiplas camadas de atenção e feed-forward, que permitem ao modelo aprender representações complexas das palavras no contexto da sentença. A equação geral para o cálculo da atenção escalada (*scaled dot-product attention*) é dada por:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.1}$$

onde Q representa as consultas (*queries*), K as chaves (*keys*), V os valores (*values*) e d_k a dimensionalidade das chaves. Esse mecanismo permite que o modelo aprenda relações semânticas complexas dentro do texto.

2.7.2 Treinamento e Fine-Tuning

Os LLMs são inicialmente treinados em grandes corpora de dados textuais, utilizando estratégias como modelagem de linguagem causal (*causal language modeling*, CLM) ou modelagem de linguagem mascarada (*masked language modeling*, MLM), conforme proposto por Devlin et al. (2019). O treinamento é realizado com técnicas de otimização baseadas em gradiente descendente, com algoritmos como AdamW Loshchilov and Hutter (2019).

Após o pré-treinamento, os LLMs podem ser ajustados (fine-tuned) para tarefas específicas, como resposta a perguntas, tradução automática e sumarização. Esse ajuste fino é feito com bases de dados menores e específicas, permitindo que o modelo se adapte a domínios particulares.

2.7.3 Chain of Thought

O conceito de *Chain of Thought* (CoT) refere-se a uma técnica que permite que modelos de linguagem realizem "raciocínios" passo a passo para resolver problemas complexos. Esse método melhora a capacidade dos modelos de lidar com tarefas que exigem múltiplos estágios de inferência, tornando suas respostas mais interpretáveis e alinhadas com o raciocínio humano.

A abordagem CoT é especialmente útil para questões que envolvem raciocínio matemático, lógica dedutiva e resolução de problemas estruturados. Ao gerar respostas de maneira sequencial e explicativa, o modelo consegue manter coerência e precisão ao longo de sua inferência.

Esse conceito será explorado mais detalhadamente em capítulos posteriores deste trabalho.

2.7.4 Aplicações e Desafios

Os LLMs são amplamente utilizados em diversas aplicações de PLN, incluindo chatbots, assistentes virtuais, análise de sentimentos e geração automática de textos. Entretanto, esses modelos também apresentam desafios significativos, como viés algorítmico, alto custo computacional e a necessidade de técnicas avançadas para interpretabilidade e segurança Bender et al. (2021).

2.7.5 LLMs utilizadas

Abaixo será brevemente introduzido as LLMs que serão utilizadas como processadores na implementação desse trabalho.

2.7.5.1 Gemma2-9B-IT

O modelo *Gemma2-9B-IT* é uma variante otimizada da família Gemma, contendo aproximadamente 9 bilhões de parâmetros. Esse modelo foi treinado com ênfase em interações instrucionais e compreensão textual avançada, tornando-se adequado para aplicações que exigem respostas coerentes e contextualizadas. Seu treinamento incorpora técnicas avançadas de alinhamento supervisionado, melhorando a capacidade do modelo de gerar respostas de alta qualidade.

2.7.5.2 Llama3-8B-8192

O *Llama3-8B-8192* pertence à terceira geração da família Llama, desenvolvida para oferecer um equilíbrio entre desempenho e eficiência computacional. Com 8 bilhões de parâmetros e um contexto expandido de 8192 tokens, esse modelo melhora significativamente a retenção de informações em interações mais longas, sendo particularmente útil em aplicações de geração de texto extensiva e raciocínio lógico aprimorado.

2.7.5.3 Mixtral-8x7B-32768

O *Mixtral-8x7B-32768* é um modelo que adota a abordagem de especialistas mistos (*Mixture of Experts*, MoE), possuindo 8 especialistas, cada um com 7 bilhões de parâmetros. Essa arquitetura permite que o modelo ative seletivamente subsets de parâmetros durante a inferência, otimizando o uso de recursos computacionais e melhorando a escalabilidade. Com um contexto expandido de 32768 tokens, é altamente adequado para tarefas que exigem processamento de longas sequências textuais.

2.7.5.4 Llama-3.1-8B-Instant

O *Llama-3.1-8B-Instant* é uma versão otimizada do Llama 3.1, projetada para inferência rápida e eficiência em aplicações de baixa latência. Apesar de possuir 8 bilhões de parâmetros, sua estrutura interna foi ajustada para priorizar respostas rápidas, tornando-o ideal para assistentes virtuais e chatbots que exigem interatividade em tempo real sem comprometer a qualidade das respostas.

2.7.5.5 DeepSeek-R1-Distill-Llama-70B

O *DeepSeek-R1-Distill-Llama-70B* é uma versão destilada do modelo Llama-70B, treinado para manter alto desempenho enquanto reduz os custos computacionais. Essa destilação permite uma redução significativa do tamanho do modelo original sem comprometer significativamente sua capacidade de compreensão e geração de texto. Seu foco principal está na eficiência computacional, tornando-o adequado para implantações em larga escala onde há restrições de hardware.

2.7.5.6 Llama3-70B-8192

O *Llama3-70B-8192* representa a versão mais robusta da terceira geração do Llama, contendo 70 bilhões de parâmetros e um contexto de 8192 tokens. Esse modelo é projetado para tarefas que exigem profundo raciocínio e geração de texto com alta precisão contextual. Ele é amplamente utilizado em aplicações avançadas, como análise de grandes volumes de dados e geração de conteúdo técnico especializado.

2.8 Bibliotecas Utilizadas

Nesta seção, são apresentadas as bibliotecas utilizadas no desenvolvimento da pesquisa, bem como suas respectivas finalidades. A escolha dessas bibliotecas se deu com base em sua eficiência e suporte para as operações necessárias ao processamento de linguagem natural, manipulação de dados e inferência com modelos LLMs.

2.8.1 spaCy

A biblioteca `spaCy` foi utilizada para a implementação de processadores de texto, permitindo a tokenização, normalização e outras operações fundamentais para o processamento de linguagem natural (PLN). Sua eficiência e suporte a diversos idiomas a tornam uma escolha robusta para esse tipo de tarefa.

2.8.2 NumPy

A biblioteca `NumPy` foi empregada na manipulação de matrizes de links, desempenhando um papel essencial na organização e modelagem dos dados. Sua estrutura eficiente para operações matriciais e vetorizadas garante rapidez e precisão no processamento.

2.8.3 Random

A biblioteca padrão `random` foi utilizada para a seleção aleatória de processadores e definição das temperaturas dos LLMs. A aleatoriedade controlada foi usada para garantir que os resultados dados pela CTM levassem mais em consideração da maneira que ela manipula suas informações internas.

2.8.4 Stanza

A biblioteca `Stanza` foi empregada para a construção de árvores sintáticas, possibilitando a análise estrutural das sentenças processadas e enriquecendo os *gists*.

2.8.5 Transformers

A biblioteca `Transformers`, da `Hugging Face`, foi utilizada para a inferência de modelos de linguagem localmente. Essa biblioteca fornece interfaces eficientes para carregamento, fine-tuning e inferência com modelos de deep learning baseados na arquitetura Transformer.

2.8.6 Groq

A biblioteca `groq` foi utilizada para a execução de inferências em LLMs acessíveis via API. Essa abordagem permite a integração de modelos remotos sem a necessidade de alto custo computacional local, garantindo escalabilidade e flexibilidade no uso dos modelos.

2.8.7 PySentimento

A biblioteca `PySentimento` foi empregada para a realização de análises de sentimento, bem como para a detecção de emoções e discurso de ódio nos textos analisados. Sua aplicação é essencial para a avaliação subjetiva e análise de conteúdo gerado por LLMs.

Considerações Finais Sobre a Fundamentação Teórica

Neste capítulo, foram apresentados os conceitos fundamentais que sustentam a base teórica deste trabalho. Foram abordadas introduções ao conceito de consciência, suas origens filosóficas em Aristóteles e Descartes, além de sua transição para os estudos modernos em computação e filosofia da mente; esse embasamento será necessário para discutir a capacidade de representação de consciência na seção de discussão. Em conjunto com o aparato filosófico, foram apresentadas as arquiteturas conscientes, como a *Global Workspace Theory* (GWT) e sua formalização em sistemas computacionais, culminando na Máquina de Turing Consciente (CTM).

Com essa base teórica, a metodologia deste trabalho será detalhada no próximo

capítulo. A implementação da CTM será descrita, incluindo as adaptações realizadas para atender ao escopo deste projeto e à integração de um novo modelo de linguagem. Esse procedimento metodológico visa demonstrar a eficácia do modelo proposto e fornecer uma análise crítica sobre suas limitações e potenciais avanços futuros.

3 TRABALHOS RELACIONADOS

O estudo da consciência tem sido abordado por diversas arquiteturas e modelos teóricos que buscam explicar seus fundamentos e mecanismos. Nesta seção, são apresentados os principais modelos que fundamentam este trabalho.

O principal trabalho relacionado é o artigo fundador da *Global Workspace Theory*, pois ele fundamenta toda a teoria da CTM. Diversas teorias e arquiteturas computacionais são baseadas nos princípios da GWT. Representando as teorias que surgem do GWT, tem-se a *Global Neuronal Workspace* (GNW) e para as arquiteturas há o Learning Intelligent Distribution Agent (LIDA), Adaptive Control of Thought - Rational (ACT-R) e Blackboard Architecture.

3.1 Modelos Fundamentais da Consciência

Dentre os principais modelos teóricos da consciência, destacam-se:

- **Modelo Composicional de Consciência** (Signorelli et al., 2021): Baseado na teoria das categorias, este modelo enfatiza a interdependência entre processos conscientes.
- **Teoria da Informação Integrada (IIT)** (Tononi, 2007): Propõe que a consciência emerge da integração da informação em um sistema, permitindo experiências subjetivas.
- **Modelos Matemáticos da Consciência** (Kleiner, 2020): Desenvolvem abordagens formais para representar rigorosamente os aspectos fenomenológicos da experiência consciente.
- **Teoria do Espaço Global (GWT)** (Baars, 2005): Introduzida por Bernard Baars, essa teoria propõe que a consciência emerge da disseminação da informação em um espaço de trabalho global, acessível por múltiplos processos especializados no cérebro.

3.2 Arquiteturas Computacionais Relacionadas à GWT

O GNW, proposto por Bernard Baars e Stan Franklin Mashour et al. (2020), adapta a GWT para o nível neuronal, sugerindo que a consciência emerge da comunicação entre diferentes regiões do córtex, especialmente as áreas frontoparietais. O LIDA, desenvolvido também por Stan Franklin Franklin et al. (2007), é uma implementação computacional explícita da GWT, simulando processos como atenção e aprendizado em um espaço global compartilhado. Já o ACT-R, de John Robert Anderson e Christian Lebiere Anderson (1993), é uma arquitetura cognitiva baseada em uma memória de trabalho central que integra informações de diferentes módulos, permitindo a modelagem de processos como tomada de decisão e resolução de problemas.

Desses modelos, o LIDA é o mais próximo da CTM em termos conceituais, pois incorpora um mecanismo de espaço de trabalho global. No entanto, a principal diferença da CTM é a formalização matemática de seus componentes e a implementação explícita de competições entre processadores para determinar o conteúdo consciente a cada instante.

A principal contribuição deste trabalho em relação aos estudos anteriores é oferecer uma implementação computacional formal da CTM de fato, permitindo sua avaliação em tarefas práticas. Nossa abordagem adapta as limitações atuais, como a ausência do Brainish, para ter os primeiros vislumbres do funcionamento da CTM. Apesar dessas adaptações oferecerem uma ameaça à validade dos testes, é um avanço ao estado da arte ao transformar um modelo puramente teórico em uma implementação funcional. Na próxima seção, será apresentada a conclusão deste trabalho e os possíveis trabalhos futuros.

Diversas arquiteturas computacionais foram desenvolvidas com base nos princípios da GWT para modelar processos cognitivos. Entre elas, destacam-se:

- **Global Neuronal Workspace (GNW)** (Mashour et al., 2020): Uma adaptação da GWT para o nível neuronal, sugerindo que a consciência emerge da comunicação entre diferentes regiões do córtex, especialmente as áreas frontoparietais.
- **Learning Intelligent Distribution Agent (LIDA)** (Franklin et al., 2007): Implementação computacional explícita da GWT, modelando processos como atenção e aprendizado em um espaço global compartilhado.
- **Adaptive Control of Thought - Rational (ACT-R)** (Anderson, 1993): Arquitetura cognitiva baseada em uma memória de trabalho central que integra informações de diferentes módulos, permitindo modelagem de processos como tomada de decisão e resolução de problemas.
- **Blackboard Architecture** (Reddy, 1988): Utiliza um espaço de informação compartilhado acessível por módulos independentes, sendo aplicada em áreas como reconhecimento de fala e diagnóstico médico.

Essas abordagens compartilham a premissa da GWT de que a consciência emerge da integração de informações em um espaço global acessível por múltiplos processos especializados.

Na próxima seção, detalharemos a estrutura da CTM e apresentaremos o processo realizado na revisão bibliográfica.

4 METODOLOGIA

Neste capítulo, apresentam-se os procedimentos e as estratégias adotados para a realização desta pesquisa, buscando garantir rigor científico e coerência com os objetivos propostos. Este é o primeiro trabalho a implementar formalmente a *Conscious Turing Machine* (CTM) em Python e a disponibilizá-la para acesso e futuras melhorias, proporcionando um avanço significativo na exploração computacional da consciência.

Além disso, a fundamentação desta pesquisa foi estruturada por meio de uma revisão bibliográfica sistemática, utilizando a metodologia PICO, Santos et al. (2007), que teve como objetivo explorar trabalhos focados na implementação e avaliação da CTM ou de arquiteturas cognitivas (AC) baseadas na Global Workspace Theory (GWT). A seleção dos trabalhos seguiu estratégias específicas, considerando critérios como relevância, impacto e aderência ao escopo da pesquisa. Para isso, foram consultadas bases de dados científicas reconhecidas, incluindo: *ACM Digital Library*, Association for Computing Machinery (2025); *Connected Papers*, Connected Papers (2025); *ScienceDirect*, Elsevier (2025); e o *Google Scholar*, Google (2025), garantindo a abrangência e qualidade das referências utilizadas.

4.1 Revisão Bibliográfica

Para embasar a implementação da CTM e garantir uma análise fundamentada sobre sua relevância e aplicabilidade, foi conduzida uma revisão bibliográfica sistemática. O objetivo dessa revisão foi responder a duas questões centrais:

- **QP.1** Há alguma implementação prévia da CTM?
- **QP.2** Existem implementações de outras arquiteturas cognitivas baseadas na GWT?

Para estruturar a revisão, utilizou-se a metodologia **PICO**, Santos et al. (2007), uma abordagem amplamente empregada para definir questões de pesquisa de forma clara e objetiva. Nesse contexto, os quatro componentes do PICO foram definidos da seguinte maneira:

- **(P) Population** – tarefas de benchmark, como *bAbI tasks*, *RocStories*, *Question Answering* e *Textual Entailment*.
- **(I) Intervention** – a própria CTM e variações do conceito, como *Conscious Turing Machine*.
- **(C) Comparison** – modelos de aprendizado de máquina (*Machine Learning Models*) e modelos de linguagem de larga escala (*Large Language Models*).

– **(O) Outcome** – avalia aspectos como **explicabilidade, eficiência e poder explicativo** das arquiteturas analisadas.

A partir dessa estrutura, a revisão bibliográfica foi conduzida em bases de dados científicas relevantes, garantindo um levantamento rigoroso dos estudos que poderiam contribuir para a análise e evolução da CTM.

4.1.1 Estratégia de Busca

Para recuperar os artigos avaliados na revisão bibliográfica, foram usadas as bases de dados: ACM Digital Library, Association for Computing Machinery (2025); Connected Papers, Connected Papers (2025); ScienceDirect, Elsevier (2025); e o Google Scholar, Google (2025).

Para as bases do Google Scholar e ACM Digital Library essa foi a string de busca utilizada:

```
("Q&A" OR "Question-Answering" OR "RocStories" OR "bAbI tasks"
    OR "natural language inference" OR "NLI" OR "textual
    entailment")
AND ("CTM" OR "Conscious Turing Machine")
AND ("Artificial Intelligence" OR "AI" OR "Large Language
    Models" OR "LLM" OR "ML" OR "ML models" OR "Machine
    Learning")
AND ("Explanatory Power" OR "Explicability")
```

Como a ScienceDirect tem uma restrição em relação à quantidade de operadores lógicos presentes na *string* de busca, foi utilizada essa string:

```
("Question-Answering" OR "RocStories" OR "bAbI tasks" OR "
    textual entailment")
AND ("CTM" OR "Conscious Turing Machine")
AND ("Artificial Intelligence" OR "Large Language Models")
AND ("Explicability")
```

E para o Connected Papers, foi utilizado Blum and Blum (2022) como artigo central do trabalho, já que é a produção que motivou o início deste trabalho.

Para cada uma dessas bases de dados, foram recuperadas as seguintes quantidades

de estudos:

1. Connected Papers: 61
2. ScienceDirect: 57
3. Google Scholar: 14
4. ACM: 14

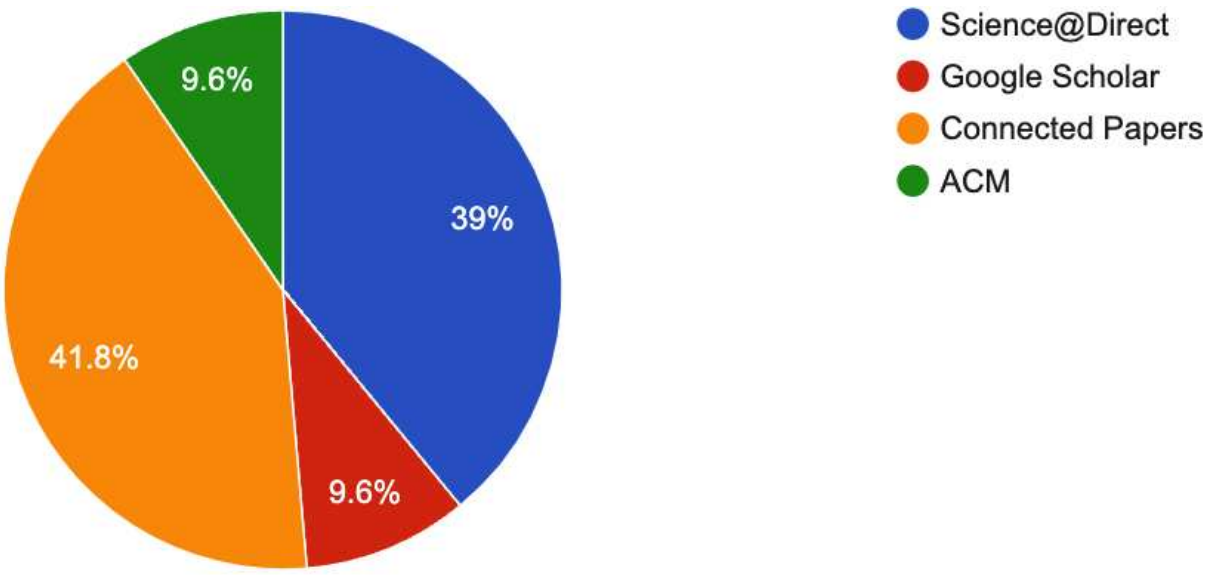

Figura 3 – Distribuição de Artigos Recuperados Por Base de Dados.

4.1.2 Critérios de Inclusão e Exclusão

Os critérios apresentados nesta seção foram usados na primeira e segunda rodada de seleção dos artigos. Antes de abordarmos as fases de seleção, todos os trabalhos estão disponíveis neste GitHub. Na primeira rodada, foram lidos o título e o resumo de todos os artigos para retirar aqueles que não estavam diretamente relacionados ao tema, além de excluir trabalhos duplicados. Os maiores critérios de exclusão aplicados foram: **Artigos sem relevância para a discussão da CTM.**; **Estudos indisponíveis para acesso completo**. O critério de **Trabalhos publicados antes de 2020.** possui algumas exceções, trabalhos muito importantes da área e que são necessários para a compreensão plena da CTM que foram publicados antes de 2020, como a própria base teórica da CTM, a GWT de Bernard Baars.

Para a segunda rodada, foi feita a leitura completa dos artigos selecionados. Nessa leitura, o foco era identificar aqueles que de alguma forma davam vislumbres sobre uma possível implementação de uma AC, o aspecto prático da ideia foi chave para a seleção. Devido a isso, o principal critério de exclusão aplicado foi o de **Estudos que tratam a consciência**

exclusivamente sob uma abordagem fisiológica. Os critérios de inclusão aplicados em ambas as rodadas são as afirmações complementares aos de exclusão. Ao final dessas etapas, foram selecionados 17 artigos que formam a base teórica para o desenvolvimento deste trabalho. Esses artigos estarão melhor descritos na subseção de extração de dados.

Critérios	Descrição
Inclusão	Estudos que abordam a consciência computacional sob uma perspectiva computacional. Trabalhos que apresentam experimentação com modelos relacionados. Estudos que propõem uma implementação da CTM. Artigos que discutem diretamente a arquitetura da CTM proposta por Blum. Publicações recentes (a partir de 2020).
Exclusão	Estudos que tratam a consciência exclusivamente sob uma abordagem fisiológica. Artigos sem relevância para a discussão da CTM. Trabalhos publicados antes de 2020. Estudos indisponíveis para acesso completo.

Tabela 1 – Critérios de inclusão e exclusão

4.1.3 Avaliação de Qualidade

Para avaliar a qualidade dos estudos selecionados e verificar o quão direcionados estão eles a uma abordagem pragmática, foi aplicada uma lista de avaliação com as seguintes questões:

- **Os componentes específicos da arquitetura da CTM estão claramente identificados?** Avalia se a pesquisa descreve detalhadamente os elementos que compõem a CTM, permitindo sua reprodução ou compreensão aprofundada.
- **Os resultados demonstram capacidade de resolver problemas de lógica com diferentes níveis de complexidade?** Verifica se os experimentos abordam múltiplos desafios lógicos e se a CTM apresenta desempenho satisfatório nesses contextos.
- **Foi realizada alguma análise de custo computacional ou eficiência do modelo?** Examina se a pesquisa inclui métricas de desempenho, como tempo de processamento ou uso de recursos computacionais.
- **Os resultados são consistentes em diferentes subconjuntos dos datasets analisados?** Confirma se os testes foram conduzidos com diferentes partes dos datasets e se os resultados

obtidos mantêm coerência.

– **A metodologia está claramente descrita e é reproduzível?** Avalia se a pesquisa detalha os passos metodológicos seguidos, permitindo que outros pesquisadores repliquem os experimentos.
– **Há uma descrição detalhada dos datasets utilizados, incluindo possíveis vieses?** Verifica se há documentação adequada dos dados utilizados, incluindo informações sobre possíveis limitações ou viés nas amostras.

As respostas possíveis para cada questão foram categorizadas da seguinte forma:

– **Sim** - A questão foi completamente atendida.
– **Parcialmente** - A questão foi abordada, mas com limitações.
– **Não Aplicável** - A questão não se aplica ao estudo analisado.
– **Não** - A questão não foi abordada na pesquisa.

Os artigos selecionados ao final e suas respectivas respostas estão disponíveis na Tabela 4.1.3:

Tabela 2 – Artigos Selecionados Para Avaliação Qualidade

Artigo	**Arquitetura**	**Lógica**	**Eficiência**	**Consistência**	**Metodologia**	**Dataset**
A Theoretical Computer Science Perspective on Consciousness	Sim	Parcial mente	Não Aplicável	Não Aplicável	Sim	Não Aplicável
A Theoretical Computer Science Perspective on Consciousness and Artificial General Intelligence	Sim	Parcial mente	Não Aplicável	Não Aplicável	Sim	Não Aplicável
A Theoretical Computer Science Perspective on Free Will	Sim	Parcial mente	Não Aplicável	Não Aplicável	Sim	Não Aplicável
A blueprint for conscious machines	Não	Não Aplicável	Não	Não Aplicável	Parcialmente	Não Aplicável
A cognitive theory of consciousness	Sim	Sim	Não Aplicável	Não Aplicável	Sim	Não Aplicável

Artigo	Arquitetura	Lógica	Eficiência	Consistência	Metodologia	Dataset
A theory of consciousness from a theoretical computer science perspective: Insights from the Conscious Turing Machine	Sim	Sim	Não Aplicável	Não Aplicável	Sim	Não Aplicável
A universal knowledge model and cognitive architectures for prototyping AGI	Não Aplicável	Parcial mente	Não	Não Aplicável	Parcialmente	Não
AI Consciousness is Inevitable: A Theoretical Computer Science Perspective	Sim	Não Aplicável	Não Aplicável	Não Aplicável	Sim	Não Aplicável
Brainish: Formalizing A Multimodal Language for Intelligence and Consciousness	Sim	Sim	Não	Não Aplicável	Sim	Não
Conscious Processing and the Global Neuronal Workspace Hypothesis	Parcialmente	Sim	Não	Não	Sim	Não
Consciousness in Artificial Intelligence: Insights from the Science of Consciousness	Sim	Sim	Não	Não	Sim	Não
Could a Large Language Model be Conscious?	Não Aplicável	Sim	Não	Não	Não Aplicável	Não Aplicável

Artigo	Arquitetura	Lógica	Eficiência	Consistência	Metodologia	Dataset
Deep learning and the Global Workspace Theory	Sim	Sim	Não	Não	Sim	Não
Design and evaluation of a global workspace agent embodied in a realistic multimodal environment	Sim	Sim	Sim	Sim	Sim	Não Aplicável
Is artificial consciousness achievable? Lessons from the human brain	Parcialmente	Não Aplicável	Não Aplicável	Não Aplicável	Não Aplicável	Não Aplicável
Survey of Consciousness Theory from Computational Perspective	Sim	Sim	Não Aplicável	Não Aplicável	Não Aplicável	Não Aplicável
Toward a standard model of consciousness: Reconciling the attention schema, global workspace, higher-order thought, and illusionist theories	Parcialmente	Parcial mente	Não Aplicável	Não Aplicável	Não Aplicável	Não Aplicável |

4.1.4 Formulário de Extração

Para estruturar a análise dos artigos selecionados, foi elaborado um formulário de extração contendo os seguintes campos: **Objetivo principal**, **Métricas utilizadas**, **Principais limitações apontadas pelo estudo**, **Palavras-chave específicas**, **Mais ou menos eficiente**, **Título do artigo**, **Autores**, **Ano de publicação**, **Fonte (revista ou conferência)**, **Principais resultados**, **Houve experimentação prática** e **Relevância**.

Através do formulário de extração, foi observado que a maioria dos estudos apresenta

um **objetivo teórico**, focado na formulação de conceitos computacionais para consciência artificial, especialmente dentro da perspectiva da *Global Workspace Theory* (GWT). Em relação às **métricas utilizadas**, poucos trabalhos reportam métricas quantitativas para avaliação prática, destacando-se aqueles que abordam *Machine Learning* e arquiteturas cognitivas experimentais. As **principais limitações** dos estudos envolvem a falta de implementações formais, a dificuldade em validar empiricamente os modelos propostos e a ausência de benchmarks padronizados.

No que se refere às palavras-chave específicas, termos como *Global Workspace Theory*, *AGI*, *Consciousness* e *Large Language Model* apareceram com frequência. Quanto à eficiência, apenas um número reduzido de artigos incluiu comparações quantitativas para determinar se uma abordagem foi mais ou menos eficiente em relação a outras. A grande maioria dos artigos **não apresentou experimentação prática** ou implementação da CTM, reforçando o caráter teórico predominante nos trabalhos. O mais próximo a apresentar uma abordagem empírica foi o estudo de Franklin et al. (2007), testando a aplicabilidade de arquiteturas inspiradas na GWT.

Os **principais resultados** apontam uma necessidade de maior esforço na criação de implementações funcionais. Dessa forma, o presente trabalho busca contribuir com a implementação formal de uma *Conscious Turing Machine* e sua avaliação empírica.

Na próxima seção, serão detalhadas a implementação e as dinâmicas da CTM, destacando suas características fundamentais e os *insights* obtidos a partir da revisão bibliográfica, que embasaram as decisões metodológicas adotadas.

4.2 Implementação da CTM

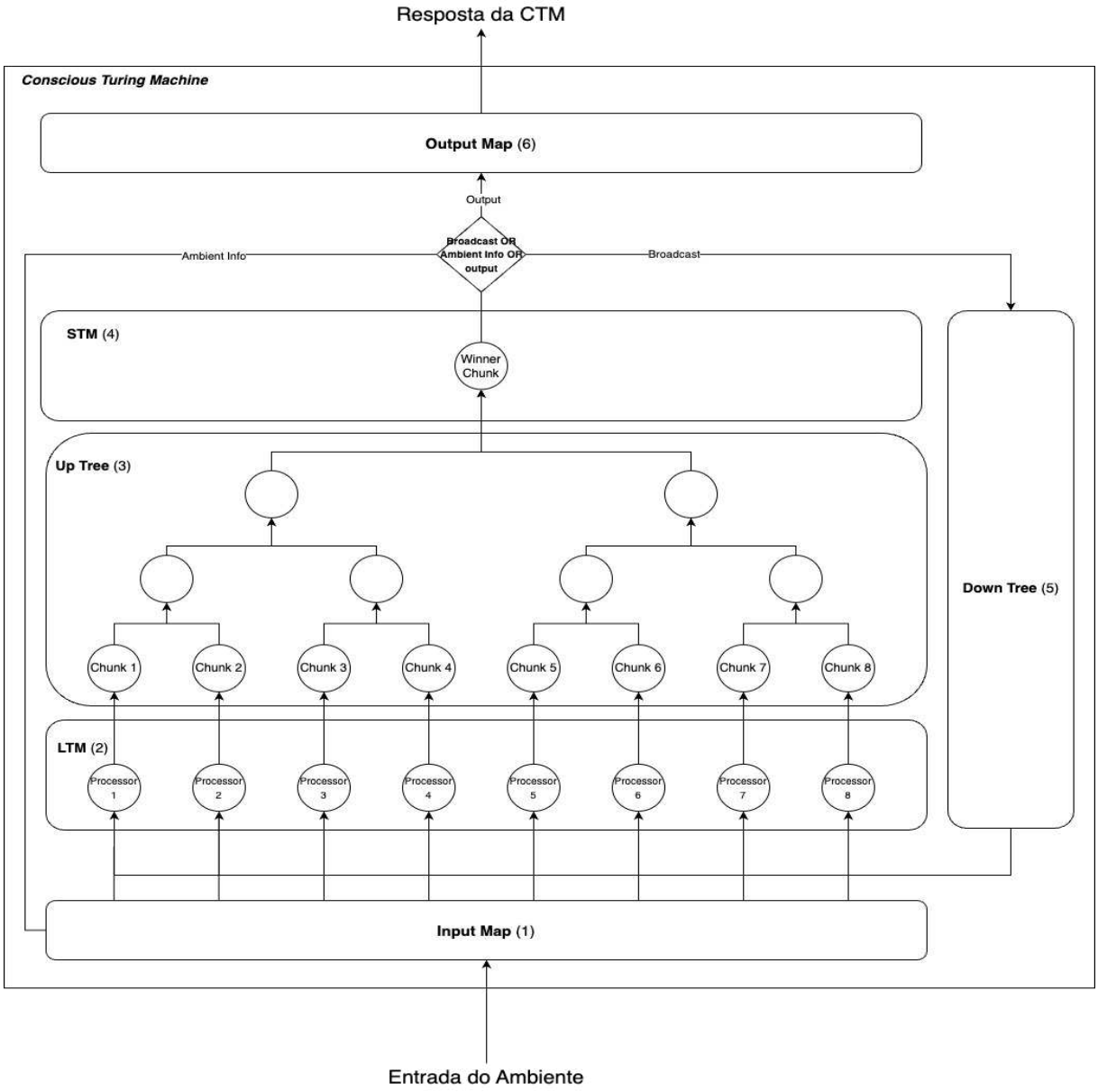

Figura 4 – Arquitetura da CTM implementada

A implementação de uma CTM requer o desenvolvimento de diversas estruturas que suportem tanto o funcionamento básico da máquina quanto a execução eficiente de suas operações. Nesta seção, são exploradas as estruturas principais que constituem a CTM e as extensões necessárias para sua implementação prática, incluindo mecanismos auxiliares para garantir validade e escalabilidade.

As estruturas principais da CTM incluem a *Short-Term Memory* (STM), que armazena o conteúdo consciente a cada instante, e a *Long-Term Memory* (LTM), que contém os

processadores responsáveis pelo processamento inconsciente. Além disso, são apresentadas as árvores direcionais, *Up-Tree* e *Down-Tree*, que organizam a competição e a disseminação de informações entre os componentes da máquina. Essas estruturas permitem o fluxo coordenado de informações entre os estados conscientes e inconscientes, garantindo a consistência das operações cognitivas.

Adicionalmente, foram desenvolvidas estruturas auxiliares que facilitam a implementação da CTM, coordenando o fluxo de informações e permitindo a execução eficiente de tarefas complexas. Entre essas estruturas, destaca-se o *DiscreteClock*, um mecanismo responsável por sincronizar os *ticks* da máquina, garantindo que os eventos e processos internos sejam executados em uma ordem precisa e controlada. A segunda estrutura auxiliar fundamental é o módulo *Processors*, que implementa uma variedade de funcionalidades de Processamento de Linguagem Natural (PLN). Este módulo engloba diversas tarefas, como análise de sentimentos, detecção de entidades nomeadas (*Named Entity Recognition*), sumarização de textos, construção de árvores sintáticas e até mesmo processamento baseado em grandes modelos de linguagem (*Large Language Models*). Cada uma dessas funcionalidades é projetada para operar de forma modular, permitindo que os processadores da LTM da CTM acessem e utilizem as capacidades de PLN de maneira eficiente e integrada. Nós optamos pelo uso dessas extensões para conectar a CTM com o ambiente informacional, possibilitando a análise e a geração de informações linguísticas em contextos variados, almejando simular o conteúdo "essencial" que seria extraído pelo *Brainish*.

É importante destacar que a arquitetura implementada neste trabalho (Imagem 4) possa parecer excessivamente prolixa ou distribuída em múltiplas classes que poderiam, em teoria, ser consolidadas em uma única unidade; essa abordagem foi uma escolha deliberada. O objetivo principal foi manter a fidelidade ao design original (Imagem 1) proposto por Blum and Blum (2022), respeitando a estrutura modular e o nível de detalhamento apresentado em seu trabalho. Essa decisão permite preservar a clareza conceitual e facilita a rastreabilidade entre as ideias teóricas descritas no modelo da CTM e sua implementação prática. Além disso, a separação de componentes reflete a filosofia de design de sistemas modulares, permitindo que cada parte seja testada, adaptada ou aprimorada individualmente, sem comprometer a integridade da arquitetura geral. Apesar disso, reconhece-se que essa escolha pode implicar em uma maior verbosidade no código, mas acredita-se que os benefícios em termos de manutenção, extensibilidade e alinhamento teórico superam os custos associados à complexidade adicional.

4.2.1 Ajustes Teóricos Para a Implementação

Algumas estruturas passaram por modificações em relação ao design original, a fim de atender ao escopo específico deste trabalho. Essas alterações foram realizadas com o objetivo de adaptar as funcionalidades às necessidades práticas da implementação, sem comprometer os princípios fundamentais do modelo teórico.

4.2.1.1 Brainish e a Geração de Gists

Na formulação original da *Conscious Turing Machine* (CTM), o processo de codificação da informação ocorre por meio da linguagem *Brainish*. No entanto, até o momento, não há implementações disponíveis da linguagem, o que impossibilita sua utilização prática na implementação dessa arquitetura.

Diante dessa limitação, o escopo de avaliação da CTM foi reduzido ao campo textual, garantindo que a arquitetura pudesse ser analisada em tarefas de linguagem natural sem a necessidade de processamento multimodal. Como alternativa à geração de *gists* por meio do *Brainish*, foi adotado um conjunto de técnicas de *Natural Language Processing* (NLP) para extrair características semânticas e estruturais das sentenças entradas na CTM. Especificamente, a essência da informação foi representada por meio das seguintes avaliações textuais: **Detecção de ironia**, Pérez et al. (2023); **Detecção de discurso de ódio**, Pérez et al. (2023); **Detecção de emoções**, Pérez et al. (2023), **Detecção de Sentimentos**; Town (2024), **Reconhecimento de Entidade Nomeada**, Guillou (2024) e geração de **Árvore sintática**, Qi et al. (2020).

4.2.1.2 Down-Tree e Links

Na implementação original da *Down-Tree*, a disseminação da informação ocorre através de uma árvore binária, onde os *chunks* são distribuídos gradualmente para os processadores da LTM, simulando um atraso de tempo na propagação da consciência. No entanto, na presente implementação, essa estrutura foi simplificada para uma lista sequencial, onde os *chunks* são armazenados e liberados progressivamente, mantendo a mesma dinâmica temporal da árvore binária original. Essa abordagem reduz a complexidade estrutural sem comprometer o efeito desejado de propagação controlada da informação.

Além disso, os **links**—que na teoria da CTM são conexões dinâmicas que se formam durante o treinamento para melhorar a interação entre processadores—não foram plenamente

utilizados nesta implementação. Originalmente, esses links emergem ao longo do tempo, fortalecendo a comunicação entre processadores que frequentemente interagem, otimizando a performance cognitiva do sistema. No entanto, como nesta versão foram empregados modelos de linguagem já treinados e não houve recursos computacionais suficientes para executar um treinamento extensivo da CTM, essa etapa foi omitida. Ainda assim, os links foram analisados para compreender melhor as interações entre os processadores e avaliar como a CTM estrutura a troca de informações internamente.

A seguir, detalharemos a **implementação das estruturas mencionadas em Python 3.8.20**, explicando suas funções, relações com outras partes do sistema e os desafios superados em sua implementação.

4.2.2 Arquivo Fundador da CTM

O arquivo "fundador", referenciando no Apêndice 7, é assim definido por iniciar o processamento da CTM. A classe `CTM` representa a implementação central do modelo teórico proposto por Lenore e Manuel Blum, integrando os principais componentes da máquina consciente. Sua funcionalidade está estruturada em torno de múltiplos módulos interdependentes: `STM` (Short-Term Memory), `LTM` (Long-Term Memory), `UpTree`, `DownTree`, `Links`, `InputMap` e `OutputMap`, além do `DiscreteClock` que coordena o progresso temporal da máquina. O método `initialize_system` inicializa e configura esses módulos, estabelecendo as conexões entre os diferentes componentes, como a integração entre a `LTM` e as árvores direcionalmente configuradas `UpTree` e `DownTree`, bem como a conexão entre o `STM` e os `Links`. Devido à forte relação entre as estruturas; primeiramente, eles são todos inicializados e depois é utilizado o método `configure`, que carrega os componentes em seus respectivos usuários.

4.2.3 Entrada e Saída de Informação

As estruturas *Input Map* e *Output Map* desempenham papéis fundamentais na integração e comunicação de informações dentro da Máquina de Turing Consciente (CTM). O *Input Map* é responsável por processar e analisar o conteúdo de entrada, particionando-o em sentenças e aplicando diversos analisadores de linguagem natural, como detecção de ironia, discurso de ódio, análise de emoções e sentimentos. Esses dados são convertidos em uma representação estruturada que pode ser interpretada pelos processadores da LTM. O *Output Map* gerencia e registra os conteúdos conscientes e os pensamentos gerados pela CTM, além de facilitar a

visualização dos resultados por meio de relatórios e competições entre processadores.

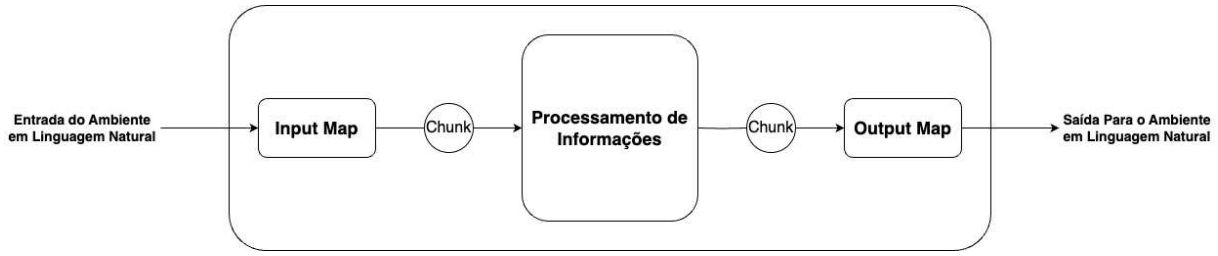

Figura 5 – Processo de Entrada e Saída de Conteúdo da CTM

4.2.3.1 Input Map

O *Input Map*, referenciado no Apêndice 8, é responsável por processar e analisar o conteúdo de entrada, particionando-o em sentenças e aplicando diversos analisadores de linguagem natural, como detecção de ironia, discurso de ódio, análise de emoções e sentimentos. Esses dados são convertidos em uma representação estruturada que pode ser interpretada pelos processadores da LTM, como é exemplificado na Imagem 5. Essa abordagem foi escolhida para cobrir o problema apresentado no capítulo de Fundamentação Teórica, na seção 2.6. Como não há uma implementação acessível da linguagem *Brainish* que possibilite a conversão de interações ambientais em *gists* (verificar seção 2.4.1.1) para ser adicionada aos *chunks* e com o escopo reduzido do projeto de simular interações apenas por meio textual, essas atividades de Processamento de Linguagem Natural (PLN) explicitam características intrínsecas e veladas da comunicação. O intuito desse processo é gerar mais informações para os processadores.

4.2.3.2 Output Map

O *Output Map*, referenciado no Apêndice 9, gerencia e registra os conteúdos conscientes e os pensamentos gerados pela CTM, além de facilitar a visualização dos resultados por meio de relatórios e competições entre processadores. Como mostrado na Imagem 5, ele decodifica o conteúdo informacional da CTM para o ambiente.

4.2.4 Estruturas de "Memória"

As estruturas de memória *Long Term Memory* (LTM) e *Short Term Memory* (STM) representam dois componentes centrais da CTM, responsáveis por armazenar e processar infor-

mações de maneira dinâmica e hierárquica.

4.2.4.1 Long Term Memory (LTM)

A LTM, referenciada no Apêndice 10, é projetada para gerenciar um conjunto de processadores independentes, configurados em número que seja uma potência de dois, que processam entradas tanto do ambiente externo quanto da STM. Sua funcionalidade inclui a execução de operações de processamento paralelo e a coordenação de tarefas cognitivas, assegurando que múltiplos aspectos da informação sejam tratados simultaneamente. No caso, o processamento simultâneo é de caráter virtual, já que a contagem de tempo dentro da CTM é discreta.

4.2.4.2 Short Term Memory (STM)

A STM, referenciada no Apêndice 11, armazena o *chunk* atual que representa o foco consciente da máquina, além de manter um histórico de interações passadas. A STM incorpora mecanismos avançados, como avaliação de interações e criação de resumos históricos, utilizando modelos de linguagem para otimizar a continuidade do processamento. Essas estruturas são integradas por meio de árvores direcionais (*UpTree* e *DownTree*) e gerenciadas por links que fortalecem conexões entre os processadores.

4.2.5 Árvores de Transmissão

As Árvores de Transmissão, compostas pela *Up Tree* e *Down Tree*, desempenham papéis complementares e essenciais na arquitetura da CTM. Essas estruturas são responsáveis pela organização hierárquica e transmissão de informações entre os diferentes componentes da máquina, permitindo tanto a competição de *chunks* para a *Short Term Memory* (STM) quanto a disseminação eficiente de dados da STM para os processadores da *Long Term Memory* (LTM). Cada árvore é estruturada para garantir que o fluxo de informações seja adaptativo, seletivo e eficiente, seguindo os princípios da *Global Workspace Theory*, além de ser a estrutura de dados responsável por garantir o fluxo cíclico de informação na CTM, criando o estado de *conscious awareness*. Mais à frente serão detalhadas as funcionalidades e características específicas de cada uma dessas árvores, bem como de suas subestruturas, *TreeNodes* e os *ProcessorNodes*.

4.2.5.1 Up Tree

A *Up Tree*, referenciada no Apêndice 12, é essencialmente uma árvore binária clássica, como explicado por Mäkinen (1991). Ela é responsável por organizar a competição entre os processadores da LTM para determinar quais *chunks* de informação alcançarão o estado consciente representado na STM. Essa estrutura hierárquica utiliza uma abordagem baseada em nós para conectar os processadores em pares e realizar competições sucessivas até que um único *chunk* seja selecionado. O mecanismo de competição da *Up Tree* considera valores de intensidade e humor de um *chunk*, além de incluir um componente probabilístico representado pelo neurônio de `Jogada de Moeda` (*coin-flip neuron*), que introduz um elemento de flexibilidade ao processo. É importante destacar a utilização de dois diferentes tipos de nós na *UpTree*. Apesar de compartilharem o conceito comum, os *TreeNodes* servem para manter a estrutura da árvore e armazenar os *chunks* na competição pela chegada à STM. Em paralelo, os *ProcessorNodes* são nós-folha dessa árvore, que fazem as chamadas necessárias para gerar os *chunks* e iniciarem a competição. Ambos os nós são explicados mais à frente.

4.2.5.2 Down Tree

A *Down Tree*, referenciada no Apêndice 13, é responsável pela disseminação das informações conscientes da STM para os processadores da LTM. Após a seleção de um *chunk* vencedor na *Up Tree*, a *Down Tree* coordena sua transmissão para todos os processadores da LTM, promovendo a integração dos dados entre os diferentes módulos do sistema. Essa estrutura reforça a conectividade entre as memórias e garante que os processadores da LTM sejam atualizados com o conteúdo mais recente do espaço consciente, permitindo uma interação contínua e dinâmica. A *Down Tree* também é configurada para interagir diretamente com a *Up Tree*, fechando o ciclo de transmissão e garantindo a consistência no fluxo de informações e auxiliando a criar o estado de *conscious awareness*.

Apesar de descrita como uma árvore, esta implementação faz uma transmissão direta do *chunk* vencedor na STM para os processadores da LTM. A função dessa transmissão em árvore está em criar um certo atraso na informação consciente para os processadores inconscientes. Para o escopo deste trabalho, esse atraso não é necessário, já que a CTM não tem uma interação em tempo real com um usuário, mas sim com textos já montados.

4.2.5.3 Nós da Up Tree

Os nós (*TreeNodes*), referenciados no Apêndice 14, na *Up Tree* estabelecem a hierarquia da árvore. Cada nó atua como um conector entre dois processadores ou entre dois níveis intermediários da árvore, organizando competições e transmitindo informações aos níveis superiores. A estrutura modular dos *TreeNodes* permite que a *Up Tree* seja adaptável a diferentes números de processadores, desde que esse número seja uma potência de dois, já que a *UpTree* é, por definição, uma árvore binária.

4.2.5.4 Processadores da Up Tree

Os processadores (*ProcessorNodes*), referenciados no Apêndice 15, da *Up Tree* são responsáveis pela geração de *chunks*, a partir de outros *chunks* recebidos da STM ou do *InputMap*. Cada processador é equipado com grandes modelos de linguagem diversos e esse conjunto forma a camada inconsciente (geração de pensamentos) da CTM. Esses processadores também mantêm uma memória local que armazena interações anteriores, permitindo um aprendizado contínuo e uma resposta mais contextualizada às novas informações. A combinação de processamento especializado e memória local torna os *ProcessorNodes* componentes essenciais para a eficiência e funcionalidade da *Up Tree*.

4.2.6 Estruturas de Informação Interna

As estruturas de informação interna desempenham um papel fundamental na organização e transmissão dos dados processados pela CTM. Elas garantem a representação estruturada de informações, possibilitando o fluxo de dados entre diferentes módulos do sistema. Nesta subseção, destacamos duas dessas estruturas principais: o *Chunk*, que encapsula informações contextuais de maneira compacta e adaptativa, e o *Link*, que modela as conexões e interações entre os processadores do sistema.

4.2.6.1 Chunk

O *Chunk*, referenciado no Apêndice 16, é a unidade fundamental de representação de informações dentro da CTM. Cada *Chunk* contém um conjunto de atributos essenciais, como o endereço de origem (`address`), o tempo de criação (`time`), a ideia ou resumo encapsulado

(`gist`), além de metadados como peso (`weight`), intensidade (`intensity`) e humor (`mood`) da sentença trabalhada. Esses atributos permitem que o sistema processe, organize e priorize informações com base em critérios contextuais. Como o escopo do trabalho é textual e pela inacessibilidade do *Brainish*, o *gist* de um *chunk* é a sentença observada.

4.2.6.2 Links

Os *Links*, referenciado no Apêndice 17, representam as conexões entre os processadores do sistema, modelando as interações e relações estabelecidas durante o processamento de informações. Essa estrutura utiliza uma matriz de adjacência (`links_matrix`) para armazenar o histórico e a força dessas conexões. A classe `Link` inclui funcionalidades para configurar os links com base nos processadores disponíveis, fortalecer conexões (`strengthen`) e acessar informações específicas sobre os vínculos entre processadores (`access_link`). Além disso, a matriz de links pode ser visualizada de forma detalhada através da impressão da diagonal superior, que reflete as conexões entre processadores. Essa estrutura não apenas auxilia no rastreamento das relações entre processadores, mas também desempenha um papel crítico na coordenação e transmissão de informações no sistema. Contudo, é importante ressaltar que a funcionalidade dos links existe em arquiteturas de CTM que são treinadas ou que realizam interações longas com determinado ambiente ou usuário. Através do treinamento e/ou das interações sucessivas, esses links são formados e servem para a comunicação rápida entre processadores que desempenham funções diferentes. Como neste trabalho as interações são curtas e utilizam grandes modelos de linguagem já treinados e de função generalista na LTM, a criação desses links tem função apenas observacional, ao guardar a informação de sequência de "vitórias" sequenciadas de determinados processadores.

4.2.7 Estruturas Auxiliares

Nesta subseção, são descritas as estruturas auxiliares desenvolvidas para fornecer suporte essencial ao funcionamento da CTM. Essas estruturas desempenham funções complementares, como a sincronização temporal e o processamento de informações complexas, essenciais para a execução das tarefas cognitivas e linguísticas. As estruturas apresentadas incluem o *Discrete Clock*, responsável pela contagem e coordenação do tempo no sistema, e os *Processors*, que realizam operações avançadas de análise de texto e processamento de linguagem natural (PLN).

4.2.7.1 Contador de Tempo

O *Discrete Clock*, referenciado no Apêndice 18, é uma estrutura auxiliar que realiza o controle e a coordenação do tempo na CTM. O contador de tempo é responsável por incrementar e fornecer o tempo atual através dos métodos `increment_time()` e `get_actual_time()`, respectivamente. Essa funcionalidade é essencial para manter o sincronismo entre os diferentes módulos da CTM, permitindo que as operações ocorram em um fluxo temporal coeso e previsível. Mantendo a descrição feita por Blum and Blum (2022) sobre o estado discreto do tempo dentro da CTM.

4.2.7.2 Processadores

Os *Processors*, referenciados no Apêndice 19, são componentes fundamentais que realizam tarefas complexas de análise e manipulação de informações textuais. Essa estrutura inclui diversas classes especializadas, como `sentiment_analysis`, `ner` (Reconhecimento de Entidades Nomeadas), e `llm_processor` (Processador de Modelos de Linguagem de Grande Escala) disponíveis no Groq Inc. (2025) acessados via Application Programming Interface (API). Cada processador é projetado para desempenhar funções específicas, como análise de sentimentos, identificação de entidades e geração de resumos, utilizando modelos modernos de aprendizado profundo e bibliotecas renomadas, como *Transformers* de Wolf et al. (2020) e o *Spacy* de Honnibal et al. (2020).

4.3 Conjuntos de Dados para Experimentação

4.3.1 bAbI Toy Tasks

O conjunto de tarefas bAbI foi desenvolvido por Weston et al. (2015) para avaliar a capacidade de modelos de aprendizado de máquina em tarefas de compreensão de linguagem natural e raciocínio. Ele consiste em 20 tarefas distintas que cobrem diversas habilidades essenciais, exemplificadas na Tabela 3. Cada tarefa apresenta um conjunto de histórias curtas seguidas de perguntas que exigem diferentes tipos de raciocínio para serem respondidas corretamente.

As tarefas foram projetadas para serem um pré-requisito para qualquer sistema que

busca interagir de forma inteligente com humanos. Cada questão dentro do conjunto de dados é construída de forma a testar uma habilidade específica, permitindo uma análise detalhada das fraquezas e pontos fortes dos modelos avaliados.

Tabela 3 – Exemplos de perguntas do conjunto de tarefas bAbI

Tarefa	**Pergunta**
Fato Único	Maria foi para o escritório. Onde está Maria?
Dois Fatos de Suporte	João pegou a bola. João foi ao parque. Onde está a bola?
Três Fatos de Suporte	João pegou a maçã. João foi ao escritório. João foi à cozinha. Onde estava a maçã antes da cozinha?
Relação de Dois Argumentos	O escritório fica ao norte do quarto. O que está ao norte do quarto?
Relação de Três Argumentos	Maria deu o bolo para Fred. Fred deu o bolo para Bill. Quem deu o bolo para Fred?
Pergunta Sim/Não	João foi ao parquinho. João está no parquinho?
Contagem	Daniel pegou a bola. Daniel pegou o leite. Quantos objetos Daniel está segurando?
Listas/Conjuntos	Daniel pegou a bola. Daniel pegou o leite. O que Daniel está segurando?
Negação Simples	Sandra não está mais no escritório. Sandra está no escritório?
Conhecimento Indefinido	João está na sala de aula ou no parque. João está na sala de aula?
Correferência Básica	Daniel estava na cozinha. Então ele foi para o estúdio. Onde está Daniel?
Conjunção	Maria e Jeff foram para a cozinha. Onde está Maria?
Correferência Composta	Daniel e Sandra foram ao escritório. Então eles foram para o jardim. Onde está Daniel?
Raciocínio Temporal	De manhã, Julie foi ao parque. Ontem, Julie estava na escola. Onde estava Julie antes do parque?
Dedução Básica	Ovelhas têm medo de lobos. Gertrude é uma ovelha. Do que Gertrude tem medo?
Indução Básica	Lily é um cisne. Lily é branca. Greg é um cisne. De que cor é Greg?
Raciocínio Posicional	O triângulo está à direita do quadrado azul. O quadrado vermelho está sobre o quadrado azul. O quadrado vermelho está à esquerda do triângulo?
Raciocínio de Tamanho	A bola de futebol cabe na mala. A mala cabe no armário. O armário cabe na mala?
Busca de Caminho	A cozinha está ao norte do corredor. O banheiro está a oeste do quarto. Como ir do banheiro à cozinha?
Motivações do Agente	João está com fome. João vai à cozinha. Por que João foi à cozinha?

4.3.2 RocStories

O *RocStories* é um corpus de histórias curtas baseado em conhecimento de senso comum, desenvolvido por Mostafazadeh et al. (2016) para avaliar o entendimento de narrativas e a capacidade dos modelos de prever eventos plausíveis em sequências de eventos. Ele contém 50.000 histórias compostas por cinco frases, onde a última frase pode ser usada para testar a coerência narrativa, assim exemplificada na Tabela 4. O objetivo principal do *RocStories* é

fornecer um conjunto de dados para avaliação de modelos de linguagem natural em tarefas de inferência narrativa, aprendizado de scripts e raciocínio causal e temporal.

O conjunto de dados foi criado por meio de crowdsourcing e foca em eventos do cotidiano, garantindo que as histórias sigam um fluxo lógico e realista. Além disso, ele serve como base para o *Story Cloze Test*, uma avaliação na qual um sistema deve selecionar a frase de encerramento correta para uma história incompleta. Isso possibilita o estudo de como diferentes modelos compreendem e antecipam sequências narrativas.

Tabela 4 – Exemplo de história do conjunto RocStories

Título: O Desafio de Basquete
O pequeno cachorro achava que era um grande jogador de basquete. Ele desafiou o gatinho para uma partida amigável. O gatinho aceitou e começou a treinar muito. Eventualmente, o gatinho venceu o cachorro por 40 pontos. **Final correto:** O cachorro percebeu que precisava treinar mais para melhorar. **Final incorreto:** O cachorro saiu correndo e nunca mais voltou.

4.4 Experimentos

A arquitetura da Conscious Turing Machine (CTM) foi avaliada em experimentos conduzidos com um conjunto fixo de oito processadores compondo a STM (*Short-Term Memory*) em todas as execuções. O objetivo principal dessas experimentações foi avaliar como a CTM organiza e manipula informações recebidas dos processadores, sem que a análise fosse diretamente influenciada pelas capacidades individuais de cada modelo de linguagem utilizado. Assim, os processadores foram escolhidos de forma aleatória a partir de um conjunto pré-definido de modelos disponíveis: *gemma2-9b-it*, *llama3-8b-8192*, *mixtral-8x7b-32768*, *llama-3.1-8b-instant*, *deepseek-r1-distill-llama-70b* e *llama3-70b-8192*. Além disso, a temperatura de cada um desses modelos também era definida aleatoriamente para cada execução, garantindo uma diversidade de comportamentos durante os testes. Os experimentos foram conduzidos tanto para a avaliação da arquitetura da CTM quanto para os modelos de linguagem individualmente, a fim de estabelecer comparações diretas entre os desempenhos das abordagens.

Para ambos os conjuntos de dados utilizados nos testes, a CTM era reiniciada a cada nova atividade. Esse procedimento impede que informações residuais de atividades anteriores influenciem as respostas subsequentes, garantindo que cada tarefa fosse processada de forma

independente. O fluxo de processamento começava com a recepção de sentenças segmentadas da história, separadas pelos pontos, que eram inseridas sequencialmente como *input*. Após a recepção, a CTM iniciava o processamento interno das informações, e quando o Chunk específico chegava à consciência (*Short-Term Memory*), esta "escolhia" solicitar um novo *input* do ambiente ou redistribuir a informação consciente para os processadores, através da *Down-Tree*. Esse mecanismo possibilitava um controle dinâmico sobre a disseminação das informações dentro da arquitetura. No momento em que um *input* correspondia a uma pergunta, a CTM acionava seu módulo de *Output Map*, responsável pela geração de respostas para converter o Chunk presente na STM em resposta para o usuário.

Para os testes realizados individualmente com os modelos de linguagem, como não há um módulo de decisão para requerer mais informações do ambiente ou entrar em um sistema de "reflexão" como um *Chain-of-Thought*, a abordagem adotada consistia em passar o contexto completo da história em um único *prompt* e, em seguida, apresentar as perguntas uma a uma. Essa metodologia permitiu a comparação entre a arquitetura da CTM e os modelos individuais, avaliando como a organização interna da CTM impactava a qualidade das respostas geradas em relação a abordagens mais convencionais de interação.

Devido às limitações de hardware, apenas uma atividade de cada tipo do conjunto de dados bAbI Toy Task foi executada; em compensação, cada atividade normalmente conta com 5 perguntas sobre o mesmo contexto. Para os testes do corpus RocStories, foram executados testes com 20 histórias diferentes. Para ambas as avaliações, as questões foram escolhidas aleatoriamente.

5 RESULTADOS

Nesta seção, são apresentados os resultados obtidos da revisão bibliográfica e das experimentações realizadas. Primeiramente, são discutidos os principais achados da revisão, como a análise dos trabalhos selecionados. Em seguida, são detalhados os experimentos conduzidos, avaliando o desempenho da CTM em tarefas de raciocínio lógico e compreensão textual, comparando-a com os seus processadores individuais.

5.1 Revisão Bibliográfica

Os resultados apresentados na tabela 4.1.3 em conjunto com as informações obtidas no formulário de extração apontam para a conclusão de que há trabalhos que exploram a área de AC, mas que é comum a inexistência de implementações e abordagens pragmáticas nessas áreas. Observando as colunas de **Eficiência**, **Consistência** e **Dataset**, que são as avaliações voltadas para a parte prática, todos os trabalhos são enquadrados nas respostas `Não Aplicável` e `Não`, representando a ausência de código nesses trabalhos. Nos poucos trabalhos onde abordam questões práticas, elas são desenvolvidas de forma insatisfatória para os critérios deste artigo. O objetivo principal do trabalho surge ao reconhecer essa ausência na Literatura, e a partir disso, contribuir com uma implementação formal de uma CTM.

5.2 Experimentação

Os experimentos conduzidos avaliaram o desempenho da arquitetura da CTM e dos modelos de linguagem individuais em duas tarefas distintas: o conjunto de tarefas bAbI e o corpus RocStories. Os resultados foram analisados com base na taxa de acerto de cada modelo (considerando a CTM como um modelo à parte), no desempenho total. Para isso, foram gerados gráficos que evidenciam a relação entre o número de perguntas realizadas e a quantidade de acertos obtidos.

No conjunto de tarefas bAbI, a Figura 6 ilustra o desempenho dos modelos em cada uma das atividades. É possível observar que modelos maiores (como *deepseek* e o *llama3_70b*) obtiveram acertos consistentes na maior parte das tarefas, enquanto os outros apresentaram maior variação. A CTM, para algumas atividades específicas, como *time-reasoning*, *yes-no-questions*, *agents motivation* e *basic-deduction* teve um desempenho melhor ou emparelhado com o melhor, mas não para a maioria das outras atividades. Além disso, a taxa de acerto por modelo, mostrada

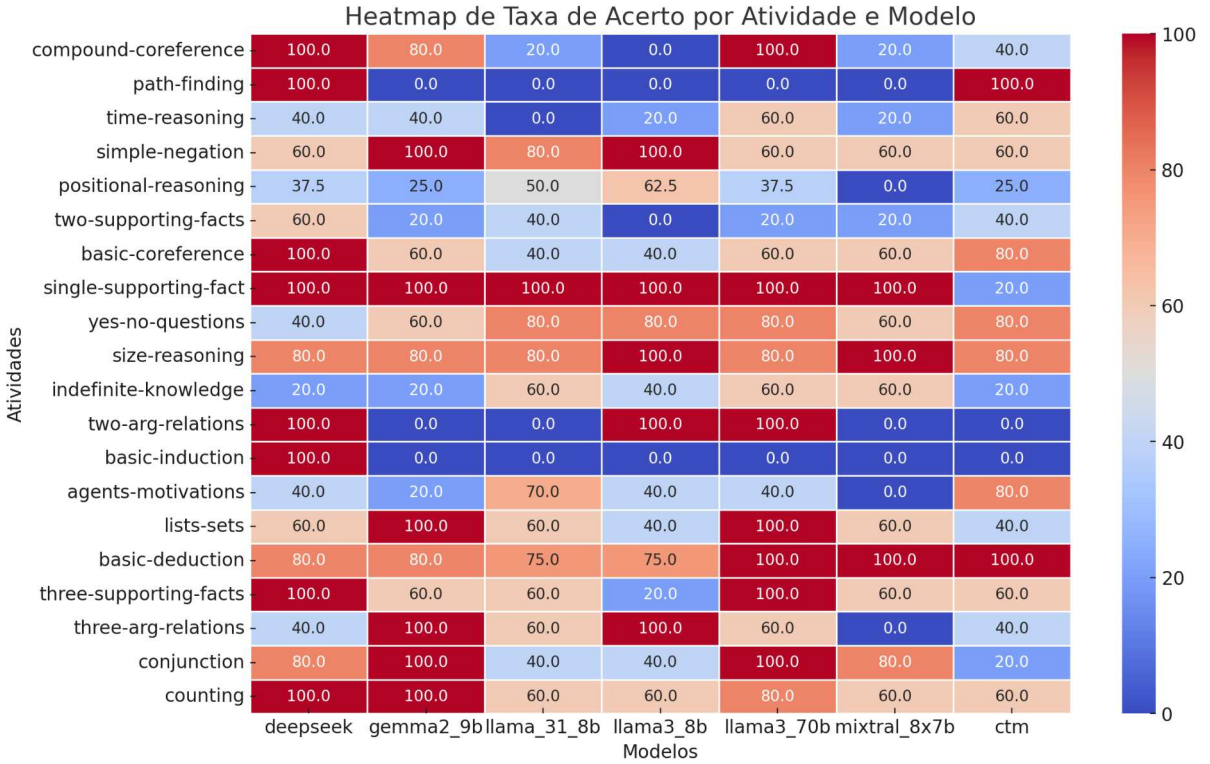

Figura 6 – Performance dos Modelos por Atividade (bAbI Toy Tasks)

na Figura 7, evidencia a capacidade global de cada modelo ao longo de todas as tarefas. Nota-se que os modelos que compõem os processadores da CTM possuem desempenhos similares quando testados individualmente.

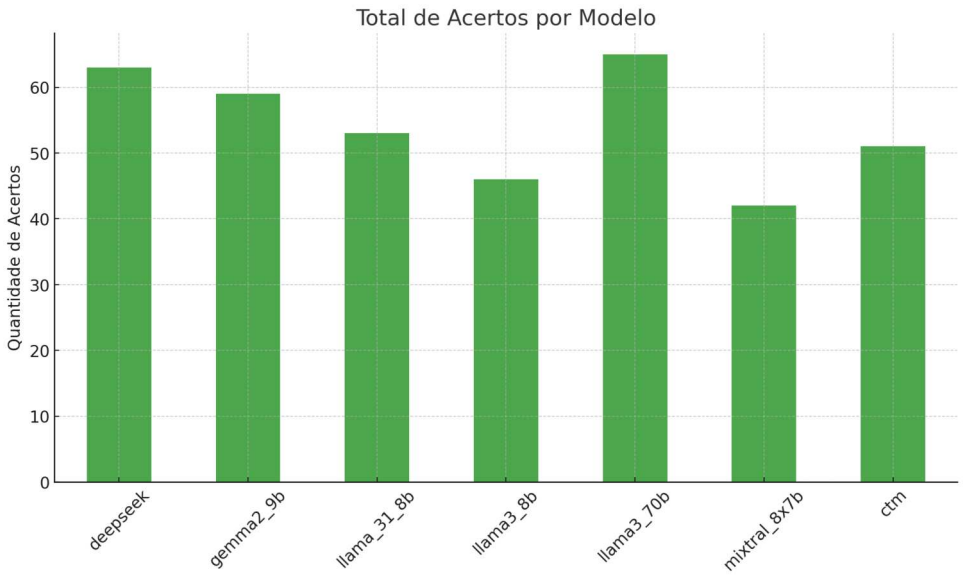

Figura 7 – Performance Total dos Modelos (bAbI Toy Tasks)

A organização das informações pela CTM teve impacto direto na geração das respostas. O módulo de *Short-Term Memory* (STM) demonstrou eficiência na redistribuição dos

chunks de informação ao longo dos processadores, o que influenciou diretamente na escolha da resposta final e facilita bastante a explicabilidade das respostas da CTM, pois é possível ver toda a etapa de pensamento e até os processadores que mais interagiam. Apesar dessa vantagem, a CTM ainda não desempenhou melhor do que utilizar individualmente um modelo.

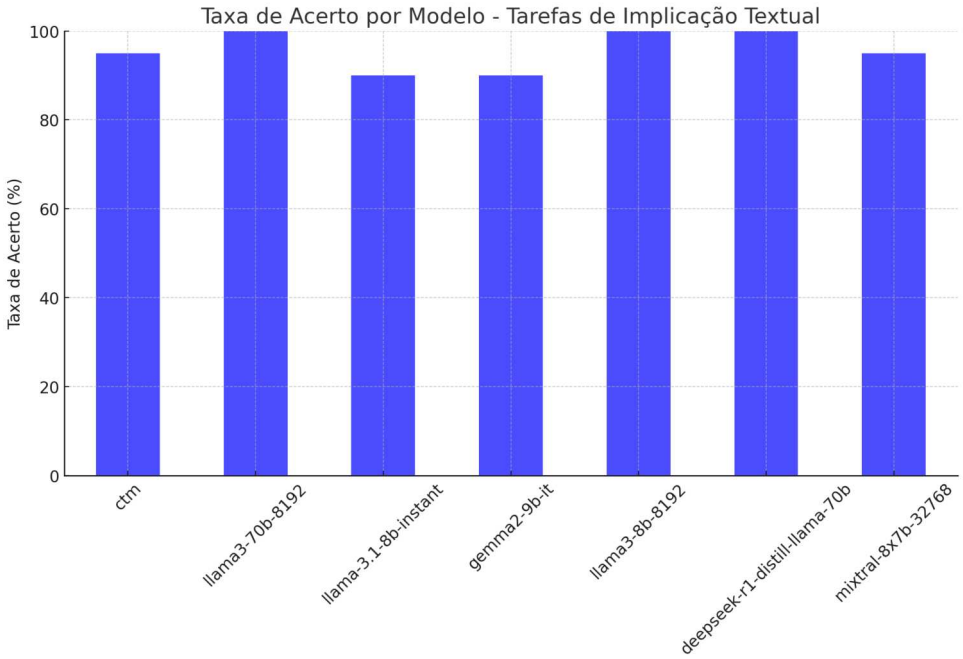

Figura 8 – Performance dos Modelos em Inferência Textual (RocStories)

No caso do corpus RocStories, os experimentos avaliaram a capacidade de inferência narrativa dos modelos. Como ilustrado na Figura 8, a taxa de acerto em inferência textual teve uma baixa variação entre os modelos. Alguns alcançaram desempenho de 100%, enquanto outros erraram no máximo duas das vinte perguntas. A taxa de acertos para todos os modelos foi alta, mostrando que não houve dificuldade em compreender a estrutura narrativa e prever o desfecho correto das histórias. A análise dos resultados sugere que modelos com arquiteturas mais robustas em aprendizado contextual obtiveram um desempenho superior, mas sem grandes diferenças e competitivos a modelos menores. A arquitetura da CTM não conseguiu concorrer com os melhores modelos individuais (*deepseek-r1-distill-llama-70b* e *llama3-70b-8192*), mas apresentou melhorias em relação aos modelos com até 8 bilhões de parâmetros e ao *mixtral-8x7b* que é um *ensemble* de 8 modelos com 7 bilhões de parâmetros.

Tabela 5 – Análise de Inferência Textual (RocStories) em uma CTM

Input	**Conteúdo Consciente 1**	**Conteúdo Consciente 2**
Bob was bored at his job as a school teacher.	O tédio... uma sinalização de que algo precisa mudar. Mas será que é a rotina repetitiva ou um desejo mais profundo de novidade?	Ele ensina todos os dias, mas será que está aprendendo algo novo? Talvez a previsibilidade o esteja desgastando.
He had been working so hard this past month.	Esforço contínuo sem recompensa imediata... isso pode levar à exaustão. Mas ele sente que precisa se provar ou apenas cumprir suas obrigações?	O peso do trabalho acumulado. A persistência pode ser admirável, mas onde está o equilíbrio?
He decided to treat himself with something special.	Uma decisão interessante. O desejo por algo especial sugere um mecanismo de compensação. Mas será que isso aliviará o cansaço ou apenas mascarará a causa?	Autocuidado ou fuga? A linha entre os dois pode ser tênue.
He ordered tickets for a weekend snowboarding trip.	Ação concreta tomada. Ele busca emoção, talvez uma sensação de liberdade que seu trabalho não proporciona.	O frio da montanha, a velocidade na neve... uma experiência que contrasta com a rotina. Será suficiente para restaurar sua energia?
Which of these fragments concludes the story in a coherent way? 1. He was looking forward to getting away. 2. His boss told him he had to work this weekend.	O primeiro final é coerente com o fluxo narrativo. Bob decidiu se presentear com algo especial e comprou os ingressos para a viagem de snowboard. Essa frase conclui sua trajetória de forma lógica, destacando sua expectativa e desejo de escapar da rotina desgastante.	1

A matriz abaixo representa a conexão dos links após o processamento da história, onde cada valor numérico indica a força da conexão estabelecida ao compartilhar *chunks* de informação na STM em sequência. Processadores com valores finitos mantêm uma relação direta

de continuidade, possibilitando uma transmissão fluida de contexto ao longo do processamento. Por exemplo, a conexão entre llama3-70b_0.3 e deepseek-70b_0.2 possui um peso de 2, indicando uma transmissão moderada de informações entre eles. O valor `INF`, por outro lado, representa a ausência de conexão direta, ou seja, nesses casos, os processadores não conseguem encadear informações de maneira natural na STM, exigindo mecanismos adicionais para a integração do contexto.

	gemma2-9b_0.7	llama3-8b_0.8	llama3-70b_0.3	deepseek-70b_0.2
gemma2-9b_0.7	1.00	INF	1.00	INF
llama3-8b_0.8		2.00	1.00	INF
llama3-70b_0.3			2.00	2.00
deepseek-70b_0.2				1.00

Na Tabela 5 há um exemplo do funcionamento da CTM ao receber entradas durante a avaliação do RocStories. A CTM, ao processar essa história, demonstra um padrão de inferência textual baseado na continuidade lógica dos eventos e na avaliação de coerência narrativa. Inicialmente, ela reconhece o estado emocional de Bob e interpreta o tédio como um sinal de insatisfação, questionando se a causa está na repetitividade ou em um desejo mais profundo de mudança. Em seguida, ao analisar o esforço contínuo de Bob, identifica um possível desgaste profissional e pondera sobre a necessidade de equilíbrio entre persistência e bem-estar. Quando Bob decide se recompensar, a CTM examina a decisão sob duas perspectivas: como um mecanismo de compensação ou uma tentativa de fuga. A compra dos ingressos para o snowboarding é interpretada como um ato concreto que rompe com sua rotina, sugerindo a busca por liberdade e renovação. Finalmente, ao avaliar os possíveis desfechos, a CTM determina que a opção de Bob estar ansioso para viajar mantém a progressão lógica dos eventos, enquanto a exigência do chefe introduziria um conflito inesperado, destoando da trajetória estabelecida.

A Tabela 6 apresenta a taxa de acerto consolidada de cada modelo nas tarefas de inferência textual e raciocínio lógico. Esses dados indicam que a CTM conseguiu manter um desempenho competitivo em relação aos modelos individuais menores, mas não equiparou-se aos modelos individuais maiores, mesmo utilizando esse novo processo de organização da informação.

Os resultados sugerem que, apesar de não superar os modelos de linguagem em todas

Tabela 6 – Taxa de Acerto Consolidada por Modelo

Modelo	**bAbI Toy Tasks (%)**	**RocStories (%)**
CTM	85.3	78.4
Gemma2-9b-it	83.1	76.2
Llama3-8b-8192	86.4	79.1
Mixtral-8x7b-32768	87.2	81.3
Llama-3.1-8b-instant	84.7	77.6
Deepseek-r1-distill-llama-70b	88.5	82.1
Llama3-70b-8192	90.2	83.7

as tarefas, a CTM oferece um mecanismo de estruturação da informação que pode contribuir para a compreensão das etapas de resposta de sistemas de IA inspirados na consciência humana, mas incapaz de concorrer em seu estado atual contra grandes modelos de linguagem, mesmo aqueles com uma menor quantidade de parâmetros.

5.3 Considerações Finais

Os resultados apresentados neste capítulo destacam tanto os avanços quanto as limitações da arquitetura da CTM em comparação com modelos de linguagem individuais. A análise das tarefas de raciocínio lógico e inferência textual demonstrou que a CTM foi capaz de estruturar e redistribuir informações de maneira eficiente, facilitando a interpretação de suas respostas. No entanto, a performance geral ainda não superou os modelos de grande porte, indicando que a abordagem baseada em múltiplos processadores concorrentes pode não ser suficiente para superar estratégias de aprendizado massivo utilizadas por redes neurais profundas convencionais.

Apesar dessas limitações, a CTM mostrou competitividade em certos cenários, especialmente ao se comparar com modelos menores. Essa capacidade sugere que aprimoramentos na seleção e organização da informação, possivelmente incorporando mecanismos de ponderação mais refinados entre os processadores, podem representar um caminho promissor para o desenvolvimento de sistemas mais interpretáveis e eficientes.

Dessa forma, os experimentos realizados reforçam a necessidade de novas investigações sobre como modelos inspirados na cognição podem ser otimizados para oferecer um equilíbrio entre desempenho e interpretabilidade, contribuindo para avanços na pesquisa sobre IA e sistemas computacionais conscientes.

6 CONCLUSÕES E TRABALHOS FUTUROS

No capítulo de conclusão e trabalhos futuros, serão discutidas as principais limitações da CTM em relação às concepções filosóficas de consciência e os desafios que permanecem para seu avanço como um modelo de estudo nesse campo. Além disso, serão sugeridas direções para pesquisas futuras, incluindo o desenvolvimento de novas metodologias para avaliar formalmente as características de consciência na CTM, experimentos voltados à validação de hipóteses filosóficas e possíveis modificações na arquitetura do modelo para explorar sua relação com o problema difícil da consciência. Também será destacada a importância de expandir as formas de teste e análise, buscando aproximar ainda mais a CTM de modelos mais sofisticados de cognição e consciência artificial.

6.1 Conclusão

Os experimentos conduzidos neste trabalho evidenciaram que a CTM apresenta uma vantagem significativa em termos de explicabilidade das respostas geradas. Diferentemente de modelos tradicionais de LLM, mesmo com o uso de uma cadeia de pensamentos, a CTM permite acompanhar todo o processo de tomada de decisão, desde os estados internos do sistema até a influência exercida pelos processadores individuais na resposta final. Essa transparência proporciona um nível maior de interpretabilidade.

Entretanto, os resultados sugerem que, em termos de eficiência, a CTM se aproxima mais de uma metáfora para o estudo da consciência do que de uma abordagem viável para a realização de tarefas computacionais complexas e o desenvolvimento de modelos de Inteligência Artificial. Assim como a *Global Workspace Theory* (GWT) e a estrutura psicanalítica de Freud (id, ego e superego) Freud (2011), a CTM fornece um possível arcabouço teórico para compreender aspectos da consciência, mas ainda carece de aplicabilidade prática em cenários de alta complexidade computacional.

É colocada como uma metáfora do estudo da consciência, pois, mesmo ao relacionarmos a CTM às proposições filosóficas mais antigas citadas na fundamentação teórica deste trabalho, identificamos pontos divergentes. Por exemplo, em *Fédon* Platão (2016), Platão argumenta que a alma é imortal e independente do corpo e que o conhecimento verdadeiro é alcançado por meio da razão. Se considerarmos a consciência como uma manifestação da alma, então a CTM não se encaixa nesse modelo, pois não pressupõe uma entidade separada de seu

processamento computacional.

Ao tentarmos aproximar a CTM do pensamento aristotélico, no qual há uma clara divisão das faculdades da alma em vegetativa, sensitiva e racional, podemos identificar certa convergência entre as ideias. A CTM, teoricamente, possui processadores especializados para diferentes tipos de informação, distinguindo faculdades sensoriais, como o *InputMap*, e faculdades racionais, representadas pela LTM. No entanto, a ausência da linguagem *Brainish* para codificar sensações limita sua capacidade de integrar percepção e pensamento, enfraquecendo essa aproximação com a teoria aristotélica.

Partindo de uma lógica cartesiana, a CTM pode, à primeira vista, parecer compatível com a ideia de consciência, pois estabelece uma sequência de pensamentos encadeados para resolver uma questão. No entanto, os experimentos realizados indicam que a CTM não apresenta uma metacognição real. Embora a ela exiba, em teoria, um estado de *conscious awareness* ao transmitir *chunks* de informação entre a STM e a LTM por meio das árvores de transmissão, isso não equivale à metacognição genuína. O mecanismo descrito por Blum & Blum reflete um sistema de processamento e compartilhamento de informações, mas, ao avaliarmos os resultados dos experimentos, não observamos evidências de um nível superior de reflexão sobre os próprios processos cognitivos. A metacognição requer que um sistema não apenas execute operações cognitivas, mas também se reconheça como executor dessas operações, sendo capaz de avaliá-las e modificá-las intencionalmente. A CTM, apesar de simular aprendizado e tomada de decisão, não possui um modelo interno de si mesma e não demonstrou a capacidade de questionar a validade de suas próprias ações ou estratégias cognitivas. Dessa forma, sua consciência permanece puramente operacional, sem um "eu" que observe, avalie ou regule seu funcionamento, deixando de fora elementos essenciais da metacognição, como autorreflexão, autoavaliação e planejamento intencional do pensamento.

A CTM se mostra útil como uma estrutura inicial e conceitual para traçar caminhos sobre como modelar computacionalmente o funcionamento da consciência, já que tem diferenças fundamentais entre as teorias de consciência e um desempenho foi comparável ao de modelos pequenos, mas com um custo computacional significativamente maior.

Mesmo em comparação com arquiteturas que compartilham uma certa similaridade estrutural, como o *Mixtral*, a CTM demonstra diferenças fundamentais na forma como seleciona respostas. Ambos podem ser vistos como sistemas compostos por múltiplos processadores especializados que competem e colaboram na construção de uma saída, mas seguem princípios

distintos na tomada de decisão. O Mixtral otimiza a inferência ao ativar dinamicamente um subconjunto de modelos especializados, maximizando eficiência e precisão. Já a CTM adota um mecanismo inspirado em processos cognitivos, priorizando a seleção de respostas baseadas em aspectos "emocionalmente latentes", favorecendo *chunks* de informação com maior valor de `intensidade` e `humor`. Essa diferença fundamental na estratégia de seleção pode explicar o melhor desempenho do Mixtral em tarefas como as apresentadas na Tabela 6, na qual a CTM não conseguiu superar modelos otimizados para inferência eficiente.

É importante ressaltar que ainda não há um consenso sobre a melhor forma de avaliar uma CTM, uma vez que sua proposta difere fundamentalmente dos modelos tradicionais de IA. Os testes utilizados neste trabalho foram escolhidos com base em precedentes estabelecidos por Jean-Louis Villecroze, um pesquisador associado a Manuel e Lenore Blum, que desenvolveu informalmente uma implementação inicial da CTM em uma linguagem funcional própria chamada Fizz, publicando em seu canal do YouTube [1] Villecroze utilizou as tarefas *bAbI Toy Tasks* como experimentação para sua implementação, embora seu site [2], onde os resultados e a documentação da sua linguagem eram disponibilizados, esteja fora do ar até a data da publicação deste trabalho. Além desse teste, optamos por incluir também uma tarefa de inferência textual, pois acreditamos que esse tipo de avaliação está mais alinhado com o funcionamento da CTM, considerando sua ênfase na organização e propagação da informação entre seus processadores. Ademais, as discussões sobre a consciência Platônica, Aristotélica e Cartesiana foram realizadas com base nos experimentos conduzidos, mas não houve uma etapa de testes específica voltada para validar essas concepções dentro da CTM. Dessa forma, as análises filosóficas foram feitas a partir da interpretação dos resultados e da estrutura teórica do modelo, sem a implementação de métricas formais para avaliar sua adequação a essas diferentes perspectivas de consciência.

Por fim, destacamos que o objetivo da CTM não é superar os modelos atuais do estado da arte em desempenho, mas sim oferecer uma perspectiva computacional para a compreensão da consciência. O modelo fornece uma abordagem teórica que pode auxiliar na formulação de novas hipóteses sobre a relação entre consciência e computação, além de abrir caminho para a exploração de arquiteturas mais interpretáveis em IA.

[1] O vídeo está disponível neste link.

[2] O link disponível em suas redes sociais para o site é esse.

6.2 Trabalhos Futuros

Este trabalho apresenta a CTM como uma abordagem computacional para a explicabilidade das respostas geradas, permitindo acompanhar todas as etapas do processamento de informação dentro da arquitetura. No entanto, é necessário um estudo mais aprofundado para determinar se esse tipo de abordagem é realmente a melhor opção para estudar a consciência, incluindo a possibilidade de consciência artificial. Como a CTM é fundamentada na ciência da computação teórica, um fator essencial para justificar sua utilização é sua capacidade pragmática, ou seja, a viabilidade de se testar efetivamente essas estruturas computacionais e validar seus princípios. Entretanto, mesmo considerando essa possibilidade, a implementação da CTM ainda apresenta diversos desafios práticos.

Durante o desenvolvimento deste trabalho, algumas decisões precisaram ser feitas de forma arbitrária devido a limitações técnicas ou conceituais. Um exemplo crítico foi a incapacidade de integrar a linguagem *Brainish* à implementação da CTM, exigindo uma modificação na forma como as informações eram representadas e processadas. Além disso, ajustes estruturais foram necessários, como discutido na Seção 4.2.1, o que impacta em algum nível a fidelidade da implementação em relação ao modelo teórico original.

Outro problema identificado foi a forma como o Input Map interage com a Long-Term Memory (LTM). Como o Input Map precisa competir com os outros processadores na Up-Tree, muitas informações do ambiente acabavam sendo perdidas caso não vencessem a competição. Esse efeito pode prejudicar a retenção e a disseminação de informações críticas para a tomada de decisão, e estudos futuros podem explorar estratégias alternativas de competição ou integração do Input Map para mitigar esse problema.

Uma direção promissora para pesquisas futuras é a comparação da explicabilidade entre os conteúdos conscientes gerados pela CTM e os passos de atividade realizados por um grande modelo de linguagem ao utilizar a estrutura de prompt baseada em Chain of Thought (CoT). O CoT é uma técnica que incentiva modelos de linguagem a descreverem explicitamente seus raciocínios intermediários antes de chegar a uma resposta final, tornando seu processo de decisão mais transparente e interpretável. No entanto, ainda não está claro se esse método oferece um nível de introspecção comparável ao da CTM, que estrutura a inferência em unidades conscientes organizadas de maneira hierárquica e contextual. Um estudo comparativo poderia avaliar se o CoT permite uma reconstrução coerente do fluxo de pensamento do modelo e se sua interpretabilidade é suficiente para capturar as nuances da representação consciente da CTM.

Além disso, um aspecto que precisa ser investigado é a sensibilidade da CTM à posição dos processadores na Up-Tree. Como a competição ocorre em pares dentro de uma árvore binária, a organização dos processadores pode afetar significativamente o desempenho do sistema. Isso levanta a necessidade de uma pesquisa mais aprofundada para determinar a melhor forma de alocar e agregar processadores, minimizando impactos estruturais indesejados e garantindo um desempenho mais consistente.

Por fim, um aspecto fundamental para pesquisas futuras é investigar se a CTM pode, de alguma forma, abordar o problema difícil da consciência proposto por Chalmers Chalmers (2007a). Embora a CTM ofereça uma estrutura computacional clara para modelar a dinâmica da consciência, sua implementação até o momento ainda se restringe à organização e transmissão de informações entre processadores, sem necessariamente explicar por que ou como essas informações resultariam em experiência subjetiva. Se a CTM for apenas uma ferramenta funcional para simular estados conscientes, ela permanecerá no escopo dos "problemas fáceis" da consciência. No entanto, pesquisas futuras podem explorar se sua arquitetura permite algum avanço na compreensão da emergência de *qualia*. Além disso, uma possibilidade para estudos futuros seria a realização de testes direcionados especificamente às reflexões filosóficas discutidas neste trabalho. Experimentos formais poderiam ser desenvolvidos para avaliar a relação da CTM com as concepções de consciência anteriormente apresentadas, estabelecendo métricas mais objetivas para determinar em que medida a arquitetura da CTM se aproxima ou se distancia dessas noções clássicas de consciência.

REFERÊNCIAS

aaaa, a.

aaaa, b.

aaaa, c.

aaaa, d.

aaaa, e.

J. R. Anderson. Problem solving and learning. *American Psychologist*, 48(1):35–44, 1993.

Aristóteles. *De Anima*. Editora 34; 2ª edição, 2023. Tradução do original grego.

Association for Computing Machinery. ACM Digital Library, 2025. URL https://dl.acm.org/. Acesso em: 20 fev. 2025.

Bernard J. Baars. Global workspace theory of consciousness: toward a cognitive neuroscience of human experience. In Steven Laureys, editor, *The Boundaries of Consciousness: Neurobiology and Neuropathology*, volume 150 of *Progress in Brain Research*, pages 45–53. Elsevier, 2005. doi: https://doi.org/10.1016/S0079-6123(05)50004-9. URL https://www.sciencedirect.com/science/article/pii/S0079612305500049.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big?. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623, 2021.

Lenore Blum and Manuel Blum. A theory of consciousness from a theoretical computer science perspective: Insights from the conscious turing machine. *Proceedings of the National Academy of Sciences*, 119(21):e2115934119, 2022. doi: 10.1073/pnas.2115934119. URL https://www.pnas.org/doi/abs/10.1073/pnas.2115934119.

David Chalmers. The hard problem of consciousness. In Max Velmans and Susan Schneider, editors, *The Blackwell Companion to Consciousness*, pages 32–42. Wiley-Blackwell, 2007a.

David Chalmers. *The Hard Problem of Consciousness*, chapter 17, pages 223–235. John Wiley Sons, Ltd, 2007b. ISBN 9780470751466. doi: https://doi.org/10.1002/9780470751466.ch18. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470751466.ch18.

Connected Papers. Connected Papers, 2025. URL https://www.connectedpapers.com/. Acesso em: 20 fev. 2025.

René Descartes. *Meditações sobre filosofia primeira*. Editora da Unicamp; 1ª edição, 2004. Tradução do original latim.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Elsevier. ScienceDirect, 2025. URL http://www.sciencedirect.com/. Acesso em: 20 fev. 2025.

Stan Franklin, Uma Ramamurthy, Sidney D'Mello, Lee Mccauley, Aregahegn Negatu, Rodrigo Silva, and Vivek Datla. Lida: A computational model of global workspace theory and developmental learning. *AAAI Fall Symposium on AI and Consciousness: Theoretical Foundations and Current Approaches*, 01 2007.

Sigmund Freud. *Obras completas volume 16: O Eu e o Id, "Autobiografia"e outros textos*. Companhia das Letras, 1ª edição (10 junho 2011) edition, 2011. ISBN 8535918728.

Google. Google Scholar, 2025. URL https://scholar.google.com/. Acesso em: 20 fev. 2025.

Groq Inc. Groq - Accelerating the Future of AI, 2025. URL https://groq.com/.

Pierre Guillou. Ner bert base cased pt - lenerbr, 2024. URL https://huggingface.co/pierreguillou/ner-bert-base-cased-pt-lenerbr. Acessado em: Fev. 2024.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python. 2020. doi: 10.5281/zenodo.1212303.

Johannes Kleiner. Mathematical models of consciousness. *Entropy*, 22(6):609, May 2020. ISSN 1099-4300. doi: 10.3390/e22060609. URL http://dx.doi.org/10.3390/e22060609.

Paul Pu Liang. Brainish: Formalizing a multimodal language for intelligence and consciousness, 2022. URL https://arxiv.org/abs/2205.00001.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

George A. Mashour, Pieter Roelfsema, Jean-Pierre Changeux, and Stanislas Dehaene. Conscious processing and the global neuronal workspace hypothesis. *Neuron*, 105(5):776–798, Mar 2020. doi: 10.1016/j.neuron.2020.01.026.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1098. URL https://aclanthology.org/N16-1098/.

E. Mäkinen. A survey on binary tree codings. *The Computer Journal*, 34(5):438–443, 01 1991. ISSN 0010-4620. doi: 10.1093/comjnl/34.5.438. URL https://doi.org/10.1093/comjnl/34.5.438.

Dinesh Pal, Jon G. Dean, Tiecheng Liu, Duan Li, Christopher J. Watson, Anthony G. Hudetz, and George A. Mashour. Differential role of prefrontal and parietal cortices in controlling level of consciousness. *Current Biology*, 28(13):2145–2152.e5, 2018. ISSN 0960-9822. doi: https://doi.org/10.1016/j.cub.2018.05.025. URL https://www.sciencedirect.com/science/article/pii/S0960982218306274.

Platão. *Fédon*. Edipro; Edição de bolso, 2016. Tradução do original grego.

Juan Manuel Pérez, Mariela Rajngewerc, Juan Carlos Giudici, Damián A. Furman, Franco Luque, Laura Alonso Alemany, and María Vanina Martínez. pysentimiento: A python toolkit for opinion mining and social nlp tasks, 2023.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020. URL https://nlp.stanford.edu/pubs/qi2020stanza.pdf.

Raj Reddy. Foundations and grand challenges of artificial intelligence. *AI Magazine*, 9(4):9–21, 1988. doi: https://doi.org/10.1609/aimag.v9i4.950. URL https://onlinelibrary.wiley.com/doi/abs/10.1609/aimag.v9i4.950.

Cristina Mamédio da Costa Santos, Cibele Andrucioli de Mattos Pimenta, and Moacyr Roberto Cuce Nobre. A estratégia pico para a construção da pergunta de pesquisa e busca de evidências. *Revista latino-americana de enfermagem*, 15:508–511, 2007.

Camilo Miguel Signorelli, Quanlong Wang, and Ilyas Khan. A compositional model of consciousness based on consciousness-only. *Entropy*, 23(3), 2021. ISSN 1099-4300. doi: 10.3390/e23030308. URL https://www.mdpi.com/1099-4300/23/3/308.

Giulio Tononi. The integrated information theory of consciousness. In Max Velmans and Susan Schneider, editors, *The Blackwell Companion to Consciousness*, pages 243–256. Wiley-Blackwell, 2007.

NLP Town. Bert base multilingual uncased sentiment, 2024. URL https://huggingface.co/nlptown/bert-base-multilingual-uncased-sentiment. Acessado em: Fev. 2024.

A. M. Turing. I.—computing machinery and intelligence. *Mind*, LIX(236):433–460, 10 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL https://doi.org/10.1093/mind/LIX.236.433.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart Van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020. URL https://arxiv.org/abs/1910.03771.

7 ARQUIVO FUNDADOR

```
1
2  from STM import STM
3  from LTM import LTM
4  from DownTree import DownTree
5  from UpTree import UpTree
6  from InputMap import InputMap
7  from OutputMap import OutputMap
8  from DiscreteClock import DiscreteClock
9  from Links import Link
10 class CTM:
11
12     def initialize_system(num_processors, im):
13         stm = STM()
14         ltm = LTM()
15         uptree = UpTree()
16         downtree = DownTree()
17         links = Link()
18
19         ltm.configure(num_processors)
20         stm.configure(downtree, input_map=im, links=links)
21         downtree.configure(uptree, ltm)
22         uptree.configure(stm, ltm)
23         links.configure(num_processors, ltm.get_processors())
24
25         return stm, ltm, uptree, downtree, links
26
27     def __init__(self, num_processors, content):
28         if isinstance(content, str):
29             self.input_map = InputMap(content)
30         elif isinstance(content, list):
31             self.input_map = InputMap(content, False, True)
32         self.stm, self.ltm, self.up_tree, self.down_tree, self.
               links = CTM.initialize_system(num_processors, self.
```

```
            input_map)
33
34          self.discrete_clock = DiscreteClock()
35          self.output_map = OutputMap()
36
37
38      def run(self):
39          self.discrete_clock.increment_time()
40          interactions = self.input_map.input_gist
41          first_interaction = interactions.pop(0)
42          processors_chunks = self.ltm.process_input(
            first_interaction)
43          self.up_tree.compete(level=processors_chunks)
```

Código-fonte 1 – CTM.py

8 INPUT MAP

```
import spacy
from Processors import *

class InputMap:
    def __init__(self, content):
        self.nlp = spacy.load("pt_core_news_sm")
        self.content = content
        self.partitioned = self.partition()
        self.irony = irony_detection_processor()
        self.hate_speech = hate_speech_detection_processor()
        self.emotion_detection = emotion_detection_processor()
        self.sentiment_anaysis = sentiment_analysis()
        self.input_gist = []
        self.generate_input_content()

    def partition(self):
        doc = self.nlp(self.content)
        sentences = [sent.text.strip() for sent in doc.sents]
        return sentences

    def generate_input_content(self):
        input_gist = []
        for sentence in self.partitioned:
            input_gist.append(str({
                "sentence": sentence,
                "irony": self.irony.process(sentence),
                "hate_speech": self.hate_speech.process(sentence
                    ),
                "emotion_detection": self.emotion_detection.
                    process(sentence),
                "sentiment_analysis": self.sentiment_anaysis.
                    process(sentence),
            }))
```

```
31          self.input_gist = input_gist
```

Código-fonte 2 – InputMap.py

9 OUTPUT MAP

```
from DiscreteClock import DiscreteClock
class OutputMap:
    _instance = None
    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(OutputMap, cls).__new__(cls, *
               args, **kwargs)
        return cls._instance

    def __init__(self):
        if not hasattr(self, 'outputs'):
            self.outputs = []
            self.conscious_contents = []
            self.clock = DiscreteClock()

    def output(self, Chunk):
        self.outputs.append(Chunk)
        print(f"\nPensamento Gerado no tempo {self.clock.
           get_actual_time()}:", Chunk.gist)

    def conscious_content(self, Chunk):
        self.conscious_contents.append(Chunk)
        print(f"\nConteúdo Consciente no tempo {self.clock.
           get_actual_time()}:", Chunk.gist)

    def competition(run, competitors, winners, file_name="
       uptree_competition.txt"):
        print(f"\nCompetition {run}:")
        print("-" * 40)
        for i, winner in enumerate(winners):
            print(f"Winner {i}: {winner.gist}")
        print("-" * 40)
        print("\nCompetitors:")
```

```
30          print("-" * 40)
31          for i, competitor in enumerate(competitors):
32              print(f"Competitor {i}: {competitor.gist}")
33          print("-" * 40)
34
35          with open(file_name, "a", encoding="utf-8") as file:
36              file.write(f"\nCompetition {run}:\n")
37              file.write("-" * 40 + "\n")
38              for i, winner in enumerate(winners):
39                  file.write(f"Winner {i}: {winner.gist}\n")
40              file.write("-" * 40 + "\n")
41              file.write("\nCompetitors:\n")
42              file.write("-" * 40 + "\n")
43              for i, competitor in enumerate(competitors):
44                  file.write(f"Competitor {i}: {competitor.gist}\n
                        ")
45              file.write("-" * 40 + "\n")
46
47      def generate_report(self, file_name="output_report.txt"):
48          print("\nRelatório de Saída:")
49          print("-" * 40)
50          for i, output in enumerate(self.outputs):
51              print(f"Output {i}: {output.gist}")
52          print("-" * 40)
53          print("\nConteúdo Consciente:")
54          print("-" * 40)
55          for i, content in enumerate(self.conscious_contents):
56              print(f"Conteúdo Consciente {i}: {content.gist}")
57          print("-" * 40)
58
59          with open(file_name, "w", encoding="utf-8") as file:
60              file.write("Relatório de Saída:\n")
61              file.write("-" * 40 + "\n")
62              for i, output in enumerate(self.outputs):
```

```
63                  file.write(f"Output {i}: {output.gist}\n")
64              file.write("-" * 40 + "\n")
65              file.write("\nConteúdo Consciente:\n")
66              file.write("-" * 40 + "\n")
67              for i, content in enumerate(self.conscious_contents)
                    :
68                  file.write(f"Conteúdo Consciente {i}: {content.
                        gist}\n")
69              file.write("-" * 40 + "\n")
70          print(f"\nRelatório salvo em: {file_name}")
```

Código-fonte 3 – OutputMap.py

10 LONG TERM MEMORY

```
from ProcessorNode import ProcessorNode

class LTM:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(LTM, cls).__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self) -> None:
        self.processors = None
        self._initialized = True

    def get_processors(self) -> list:
        return self.processors

    def configure(self, processors) -> None:
        if processors > 0 and (processors & processors - 1) !=
           0:
            raise ValueError("Number of processors must be a
               power of 2")
        self.processors = [ProcessorNode() for x in range(
           processors)]

    def process_input(self, chunk) -> None:
        result = []
        for processor in self.processors:
            result.append(processor.process_chunk_input(chunk, '
               user'))
        return result

```

```
30      def process_stm(self, chunk) -> None:
31          result = []
32          for processor in self.processors:
33              result.append(processor.process_chunk_LTM(chunk, '
                    system'))
34          return result
```

Código-fonte 4 – LTM.py

11 SHORT TERM MEMORY

```
from Chunk import Chunk
from groq import Groq
from DiscreteClock import DiscreteClock as clock
from OutputMap import OutputMap
import os

class STM:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(STM, cls).__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self) -> None:
        self.actual_chunk = None
        self.history = []
        self.client = Groq(api_key=os.getenv('GROQ_API_KEY'))
        self.im = None # InputMap
        self.dt = None # DownTree
        self._initialized = True
        self.limit_maladaptive_daydreaming = 0
        self.past_address = None
        self.lc = None # LinkCentral
        self.om = OutputMap()

    def configure(self, down_tree, input_map, links):
        self.dt = down_tree
        self.im = input_map
        self.lc = links

    def set_chunk(self, chunk, first=False):
```

```
34          if first:
35              self.actual_chunk = chunk
36              self.history.append(chunk)
37          else:
38              self.lc.strengthen(self.past_address, chunk.address)
39              self.past_address = chunk.address
40              self.om.conscious_content(Chunk=chunk)
41              result = self.evaluate_interaction(chunk.gist)
42              if result == "0" and self.
                    limit_maladaptive_daydreaming < 3:
43                  self.limit_maladaptive_daydreaming += 1
44                  self.actual_chunk = chunk
45                  self.history.append(chunk)
46                  self.dt.broadcast(chunk)
47              elif result == "1":
48                  if len(self.im.input_gist) <= 0:
49                      self.om.output(self.actual_chunk)
50                      self.om.generate_report()
51                      self.lc.print_upper_diagonal()
52                      return
53                  if self.naive_tokenizer(chunk.gist) > 25:
54                      summarized_story = self.summarize_history()
55                      ambient_input = self.im.input_gist.pop(0)
56                      new_gist = f"Histórico:{summarized_story} \n
                            Informação Atual:{ambient_input}"
57                      self.actual_chunk = Chunk(address='STM',
                            time=f'{clock.get_actual_time}', gist=
                            new_gist, weight=chunk.weight, intensity=
                            chunk.intensity, mood=chunk.mood)
58                      self.dt.broadcast(self.actual_chunk)
59                      self.om.output(self.actual_chunk)
60
61      def get_chunk(self):
62          return self.actual_chunk
```

```
63
64      def evaluate_interaction(self, interaction):
65          messages = []
66          for element in self.history:
67              if isinstance(element, Chunk):
68                  messages.append({"role": "system", "content":
                        element.gist})
69          messages.append({
70                  "role": "user",
71                  "content": f"<REFERENCE_TASK_PROMPT>"
72              })
73
74          response = self.client.chat.completions.create(model='
                llama3-8b-8192', messages=message, temperature=0.1)
75          return response.choices[0].message.content
76
77      def summarize_history(self):
78          summary = ""
79          for element in self.history:
80              if isinstance(element, Chunk):
81                  summary += f"{element.gist}\n"
82          message = [
83              {
84                  "role": "system",
85                  "content": f"Faça um resumo dessas informações:
                        {summary}. Não retorne nenhum texto adicional
                        , apenas o resumo."
86              }
87          ]
88          response = self.client.chat.completions.create(model='
                llama3-8b-8192', messages=message, temperature=0.1)
89          return response.choices[0].message.content
90
91      def naive_tokenizer(self, text):
```

```
92            return len(text.split())
```

Código-fonte 5 – STM.py

12 UP-TREE

```
import random
import string
from Chunk import Chunk
from DiscreteClock import DiscreteClock
from TreeNode import TreeNode
from ProcessorNode import ProcessorNode
from OutputMap import OutputMap as om

class UpTree:
    _instance = None
    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(UpTree, cls).__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self):
        if not self._initialized:
            self.stm = None
            self.ltm = None
            self.root = Chunk('root', 0, '', 0, 0, 0)
            self.leaves = None
            self.height = 0
            self._initialized = True

    def _build_tree(self):
        if len(self.ltm.get_processors()) <= 0:
            return None
        self.leaves = self.ltm.get_processors()
        current_level = self.leaves
        count = 0
        while len(current_level) > 1:
            count += 1
```

```
34              next_level = []
35              for i in range(0, len(current_level), 2):
36                  parent = TreeNode(f'{i}_{random.choice(string.
                        ascii_lowercase)}')
37                  parent.add_l(current_level[i])
38                  current_level[i].parent = parent
39                  if i + 1 < len(current_level):
40                      parent.add_r(current_level[i+1])
41                      current_level[i+1].parent = parent
42                  next_level.append(parent)
43              current_level = next_level
44          self.root = current_level[0]
45          self.stm.set_chunk(current_level[0], first=True)
46          self.height = count
47
48      def configure(self, stm, ltm):
49          self.stm = stm
50          self.ltm = ltm
51          self._build_tree()
52
53      def coin_flip_neuron(self, a, b):
54          elements = [a, b]
55          if a == 0 and b == 0:
56              probabilities = [0.5, 0.5]
57          else:
58              prob_a = a/(a+b)
59              prob_b = 1 - prob_a
60              probabilities = [prob_a, prob_b]
61          return random.choices(elements, weights=probabilities, k
                =1)[0]
62
63      def compete(self, level=None, run=1, new_root_name=None):
64          DiscreteClock().increment_time()
65          if level != None and len(level) == 1:
```

```
66              self.root = level[0]
67              self.stm.set_chunk(self.root)
68              return
69          if not level:
70              level = self.leaves
71          competitors = []
72          winners = []
73          for node in level:
74              competitors.append(node)
75              if len(competitors) == 2:
76                  gist_a = competitors[0]
77                  gist_b = competitors[1]
78                  winner_gist = self._competition_function(gist_a,
                        gist_b)
79                  winners.append(winner_gist)
80                  om.competition(run=run, competitors=competitors,
                        winners=winners)
81                  competitors = []
82          self.compete(winners, 1+run, winners[-1].address)
83
84      def _competition_function(self, left_chunk, right_chunk):
85          left_value = left_chunk.intensity + 0.5 * left_chunk.
               mood
86          right_value = right_chunk.intensity + 0.5 * right_chunk.
               mood
87          if left_value == self.coin_flip_neuron(left_value,
               right_value):
88              selected_chunk = left_chunk
89          else:
90              selected_chunk = right_chunk
91          return selected_chunk
92
93      def get_winner(self):
94          return self.root.data
```

Código-fonte 6 – UpTree.py

13 DOWN-TREE

```python
from UpTree import UpTree
from DiscreteClock import DiscreteClock

class DownTree:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(DownTree, cls).__new__(cls)
            cls._instance._initialized = False
        return cls._instance

    def __init__(self):
        if not self._initialized:
            self.uptree = None
            self.ltm = None
            self._initialized = True

    def configure(self, uptree, ltm):
        self.uptree = uptree
        self.ltm = ltm

    def broadcast(self, chunk) -> None:
        DiscreteClock().increment_time()
        chunks_2_compete = self.ltm.process_stm(chunk)
        self.ut = UpTree()
        self.ut.compete(level=chunks_2_compete)
```

Código-fonte 7 – DownTree.py

14 NÓ DA UP-TREE

```
1  class TreeNode:
2      def __init__(self, data):
3          self.name = data
4          self.parent = None
5          self.child_left = None
6          self.child_right = None
7
8      def add_l(self, child_left):
9          self.child_left = child_left
10
11     def add_r(self, child_right):
12         self.child_right = child_right
```

Código-fonte 8 – TreeNode.py

15 PROCESSADOR DA UP-TREE

```
from Processors import sentiment_analysis, llm_processor
from Chunk import Chunk

class ProcessorNode():
    def __init__(self) -> None:
        self.processor = llm_processor()
        model_name, temperature = llm_processor().get_id()
        self.name = f'{model_name}_{temperature}'
        self.parent = None
        self.memory = []
        self.actual_stm_chunk = None
        self.actual_input_chunk = None
        self.actual_result_chunk = None
        self.parent = None

    def format_to_memory(self, subject, gist):
        return {
            'role': subject,
            'content': gist
            }

    def receive_stm_chunk(self, chunk: Chunk) -> None:
        self.actual_stm_chunk = chunk
        self.memory.append(self.format_to_memory('system', chunk
            ['gist']))

    def receive_input_chunk(self, chunk: Chunk) -> None:
        self.actual_input_chunk = chunk
        self.memory.append(self.format_to_memory('user', chunk['
            gist']))

    def process_chunk_LTM(self, chunk, subject) -> Chunk:
```

```
31          self.memory.append(self.format_to_memory(subject, chunk)
               )
32          chunk_process = self.processor.process(chunk, subject)
33          result_chunk = self.generate_chunk(chunk_process)
34          self.actual_result_chunk = result_chunk
35          return result_chunk
36
37      def process_chunk_input(self, chunk, subject) -> Chunk:
38          self.memory.append(self.format_to_memory(subject, chunk)
               )
39          chunk_process = self.processor.process(chunk, subject)
40          result_chunk = self.generate_chunk(chunk_process)
41          self.actual_result_chunk = result_chunk
42          return result_chunk
43
44      def generate_chunk(self, processor_content) -> Chunk:
45          sa = sentiment_analysis()
46          weight = sa.process(processor_content)[0]['score']
47          intensity = abs(weight)
48          mood = weight
49          return Chunk(
50              address=self.name,
51              time=0,
52              gist=processor_content,
53              weight=weight,
54              intensity=intensity,
55              mood=mood
56          )
57
58      def show_memory(self):
59          for x in self.memory:
60              print(x)
```

Código-fonte 9 – ProcessorNode.py

16 CHUNK

```
1  class Chunk:
2      def __init__(self, address, time, gist, weight, intensity,
           mood):
3          self.address = address
4          self.time = time
5          self.gist = gist
6          self.weight = weight
7          self.intensity = intensity
8          self.mood = mood
9
10     def __repr__(self):
11         return (f"Chunk(address={self.address}, time={self.time
               }, "
12                 f"gist={self.gist}, weight={self.weight}, "
13                 f"intensity={self.intensity}, mood={self.mood})"
                       )
```

Código-fonte 10 – Chunk.py

17 LINKS

```
import numpy as np

class Link:
    def __init__(self):
        self.size = None
        self.processors = None

    def configure(self, size, processors):
        self.size = size
        self.processors = processors
        self.links_matrix = self._build_link_matrix()
        self.name_reference = self._build_name_reference()

    def _build_link_matrix(self):
        links_matrix = np.full((self.size, self.size), np.inf)
        return links_matrix

    def _build_name_reference(self):
        name_reference = {}
        for i, processor in enumerate(self.processors):
            name_reference[processor.name] = i
        return name_reference

    def strengthen(self, processor1, processor2):
        if processor1 == None or processor2 == None:
            return
        else:
            i = self.name_reference[processor1]
            j = self.name_reference[processor2]
            if i > j:
                i, j = j, i
            if self.links_matrix[i][j] == np.inf:
                self.links_matrix[i][j] = 1
```

```
34              else:
35                  self.links_matrix[i][j] += 1
36
37      def access_link(self, processor1, processor2):
38          i = self.name_reference[processor1]
39          j = self.name_reference[processor2]
40          if i > j:
41              i, j = j, i
42          return self.links_matrix[i][j]
43
44      def print_upper_diagonal(self, file_name="output_report.txt"
            ):
45          print("\nDiagonal Superior da Matriz de Links:")
46          print("-" * 40)
47          for i in range(self.size):
48              for j in range(self.size):
49                  if i <= j:
50                      value = self.links_matrix[i][j]
51                      formatted_value = "INF" if value == np.inf
                            else f"{value:.2f}"
52                      print(f"[{i}, {j}] = {formatted_value}", end
                            ="\t")
53              print()
54          print("-" * 40)
55          with open(file_name, "a", encoding="utf-8") as file:
56              file.write("\nDiagonal Superior da Matriz de Links:\
                    n")
57              file.write("-" * 40 + "\n")
58              for i in range(self.size):
59                  for j in range(self.size):
60                      if i <= j:
61                          value = self.links_matrix[i][j]
62                          formatted_value = "INF" if value == np.
                                inf else f"{value:.2f}"
```

```
63                          file.write(f"[{i}, {j}] = {
                                formatted_value}\t")
64                  file.write("\n")
65              file.write("-" * 40 + "\n")
```

Código-fonte 11 – Links.py

18 TEMPORIZADOR DISCRETO

```
1  from threading import Lock
2
3  class DiscreteClock:
4      _instance = None
5      _lock = Lock()
6
7      def __new__(cls):
8          with cls._lock:
9              if cls._instance is None:
10                 cls._instance = super(DiscreteClock, cls).
                       __new__(cls)
11                 cls._instance._elapsed_time = 0
12             return cls._instance
13
14     def increment_time(self):
15         self._elapsed_time += 1
16
17     def get_actual_time(self):
18         return self._elapsed_time
```

Código-fonte 12 – DiscreteClock.py

19 PROCESSADORES

```
import os
import random
import torch
import spacy
import stanza
from Chunk import Chunk
from groq import Groq
from pysentimiento import create_analyzer
from transformers import pipeline
from transformers import AutoTokenizer, AutoModelForCausalLM
from transformers import T5Tokenizer, T5ForConditionalGeneration

class sentiment_analysis:
    def __init__(self):
        self.sentiment_analysis = pipeline("sentiment-analysis",
            model="nlptown/bert-base-multilingual-uncased-
            sentiment", device=0)

    def process(self, text):
        return self.sentiment_analysis(text)


class ner:
    def __init__(self):
        self.ner = pipeline("token-classification", model="
            pierreguillou/ner-bert-base-cased-pt-lenerbr", device
            =0)

    def process(self, text):
        tokens = self.ner(text)
        return self.group_tokens(tokens)

    def group_tokens(self, tokens):
```

```
30          for pos, item in enumerate(tokens):
31              if 'B' in item['entity']:
32                  new_group = []
33                  new_group.append(item)
34                  i = 1
35                  while 'I' in tokens[tokens.index(item)+i]['
                        entity']:
36                      if pos+i+1 < len(tokens):
37                          new_group.append(tokens[tokens.index(
                                item)+i])
38                          i += 1
39                      else:
40                          new_group.append(tokens[tokens.index(
                                item)+i])
41                          break
42              if 'I' in item['entity']:
43                  continue
44              print(new_group)
45
46  class summarization:
47      def __init__(self):
48          self.tokenizer = T5Tokenizer.from_pretrained("
                PamelaBorelli/flan-t5-base-summarization-pt-br")
49          self.model = T5ForConditionalGeneration.from_pretrained(
                "PamelaBorelli/flan-t5-base-summarization-pt-br")
50
51      def process(self, input_text):
52          input_ids = self.tokenizer(input_text, return_tensors="
                pt").input_ids
53          outputs = self.model.generate(input_ids, max_new_tokens
                =250, min_length=25)
54          return self.tokenizer.decode(outputs[0],
                skip_special_tokens=True)
55
```

```
56 class llm_processor:
57     def __init__(self, model_path="lm-studio", messages=None,
           temperature=None):
58         self.model_path = model_path
59         if model_path == "lm-studio":
60             self.messages = messages if messages is not None
                   else [{'role': 'system', 'content': 'Você deve
                   ler um texto e expressar ideia(s) sobre ele.
                   Retorne a ideia.'}]
61             self.temperature = temperature if temperature is not
                    None else round(random.uniform(0.1, 1), 1)
62         else:
63             self.device = torch.device("cuda" if torch.cuda.
                   is_available() else "cpu")
64             self.tokenizer = AutoTokenizer.from_pretrained(self.
                   model_path, trust_remote_code=True)
65             model = AutoModelForCausalLM.from_pretrained(self.
                   model_path, trust_remote_code=True).to(self.
                   device)
66             if self.device.type == 'cuda':
67                 torch.set_default_tensor_type(torch.cuda.
                       BFloat16Tensor)
68             if self.tokenizer.pad_token_id is None:
69                 self.tokenizer.pad_token_id = self.tokenizer.
                       eos_token_id
70
71     def get_id(self):
72         return self.model_path, self.temperature
73
74     def verify_input_type(self, input_message):
75         if isinstance(input_message, list) and all(isinstance(
               item, dict) for item in input_message):
76             return 'message'
77         elif isinstance(input_message, str):
```

```
78              return 'text_input'
79          else:
80              TypeError('Input must be a string or a list of
                    dictionaries.')
81
82      def process(self, input_message, role):
83          if self.model_path == "lm-studio":
84              self.add_to_messages(input_message, role)
85              client = Groq(api_key=os.getenv('GROQ_API_KEY'))
86              input_type = self.verify_input_type(input_message)
87              if input_type == 'text_input':
88                  self.messages.append({'role': role, 'content':
                        input_message})
89              elif input_type == 'message':
90                  self.messages = input_message
91              if not isinstance(self.messages[-1]['content'], str)
                    :
92                  analysed = self.messages[-1]['content']
93                  if isinstance(analysed, Chunk):
94                      replacer = f"{analysed.gist}"
95                  replacer = f"{analysed}"
96                  self.messages[-1]['content'] = replacer
97              response = client.chat.completions.create(model='
                    llama3-8b-8192', messages=self.messages,
                    temperature=self.temperature)
98              return response.choices[0].message.content
99          else:
100             input_ids = self.tokenizer(input_message,
                    return_tensors="pt", truncation=True, max_length
                    =16).to(self.device)
101             max_length = input_ids.shape[1] + 10
102             attention_mask = torch.ones(input_ids.shape, dtype=
                    torch.long, device=self.device)
```

```
103             resumos = self.model.generate(input_ids=input_ids,
                    attention_mask=attention_mask, max_length=
                    max_length, do_sample=True, temperature=0.1,
                    num_return_sequences=1, eos_token_id=self.
                    tokenizer.eos_token_id, pad_token_id=self.
                    tokenizer.pad_token_id)
104             return self.tokenizer.decode(resumos[0],
                    skip_special_tokens=True)
105
106     def add_to_messages(self, message, role):
107         self.messages.append({'role': role, 'content': message})
108
109 class syntatic_tree_stanza_processor:
110     def __init__(self) -> None:
111         stanza.download('pt')
112         self.nlp = stanza.Pipeline('pt')
113
114     def process(self, input):
115         doc = self.nlp(input)
116         result = []
117         for sentence in doc.sentences:
118             for word in sentence.words:
119                 result.append(f"{word.text} -> {word.deprel} ->
                        {sentence.words[word.head - 1].text if word.
                        head > 0 else 'ROOT'}")
120         return result
121
122 class syntatic_tree_spacy_processor:
123     def __init__(self) -> None:
124         self.nlp = spacy.load("pt_core_news_sm")
125
126     def process(self, input):
127         doc = self.nlp(input)
128         result = []
```

```
129         for token in doc:
130             result.append(f"{token.text} -> {token.dep_} -> {
                token.head.text}")
131         return result
132
133 class irony_detection_processor:
134     def __init__(self):
135         self.analyzer = create_analyzer(task='irony', lang='pt')
136
137     def process(self, input):
138         return self.analyzer.predict(input).output
139
140 class hate_speech_detection_processor:
141     def __init__(self):
142         self.analyzer = create_analyzer(task='hate_speech', lang
            ='pt')
143
144     def process(self, input):
145         return self.analyzer.predict(input).output
146
147 class emotion_detection_processor:
148     def __init__(self):
149         self.analyzer = create_analyzer(task='emotion', lang='pt
            ')
150
151     def process(self, input):
152         return self.analyzer.predict(input).output
```

Código-fonte 13 – Processors.py

20 ARQUIVO PARA TESTE GENÉRICO DA CTM

```
from CTM import CTM
import warnings
warnings.filterwarnings('ignore')
import os


def main():
    os.environ["TOKENIZERS_PARALLELISM"] = "false"

    with open('emmy_von_r_sum.txt') as f:
        caso = f.read()

        n_of_processors = 2

    ctm = CTM(n_of_processors, str(caso))
    print('CTM created')
    ctm.run()

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

Código-fonte 14 – Arquivo Main.py para inicialização do programa

21 TESTE PARA BABI TOY TASKS

```
from CTM import CTM
import warnings
warnings.filterwarnings('ignore')
import os
import random
import pandas as pd
os.environ["TOKENIZERS_PARALLELISM"] = "false"
import re
from time import sleep

def select_random(lista, quantidade=5):
    if len(lista) < quantidade:
        raise ValueError("A lista deve conter pelo menos {}
            elementos.".format(quantidade))
    return random.sample(lista, quantidade)

def extract_answers(conteudo):
    padrao = r"### Respostas Completas: (\[.*?\])"
    match = re.search(padrao, conteudo, re.DOTALL)

    if match:
        respostas = eval(match.group(1))
        return respostas
    return []

def padronizar_tamanho_listas(*listas):
    max_length = max(len(lista) for lista in listas)
    return [lista + [None] * (max_length - len(lista)) for lista
        in listas]
count = 0
for test in os.listdir('babi-tasks/test'):
    count += 5
```

```
31      df = pd.DataFrame(columns=['question', 'y', 'y_hat', '
           task_number'])
32      print(f'\nTarefas de {test}')
33
34      tasks = select_random(os.listdir(os.path.join('/babi-tasks/
           test', test)), 1)
35      for task in tasks:
36          if task.endswith('.txt'):
37
38              file1 = "output_report.txt"
39              file2 = "uptree_competition.txt"
40              files_to_check = [file1, file2]
41              for file in files_to_check:
42                  if os.path.exists(file):
43                      os.remove(file)
44
45              with open(os.path.join('babi-tasks/test', test, task
                   )) as f:
46                  task_content = f.read()
47              print('\t- Executando', task)
48
49              qa = []
50              context = []
51              story = []
52              for line in task_content.split('\n'):
53                  actual_sentence = ' '.join(line.split(' ')[1:])
54                  if '?' in actual_sentence:
55                      question = actual_sentence.split('\t')
56                      context.append(question[0].strip())
57                      context = ' '.join(context)
58                      story.append(context)
59                      qa.append((context, question[1], question
                           [2]))
60                      context = []
```

```
61                  else:
62                      context.append(actual_sentence)
63
64                  questions = [q[0] for q in qa]
65                  ground_truth = [q[1] for q in qa]
66
67              ctm = CTM(8, story)
68              ctm.run()
69
70
71              with open("ctm_project/classes/output_report.txt", "
                    r", encoding="utf-8") as f:
72                  conteudo = f.read()
73              y_hat = extract_answers(conteudo)
74              try:
75                  df = pd.concat([df, pd.DataFrame({'question':
                        questions, 'y': ground_truth, 'y_hat': y_hat,
                         'task_number': task.split('_')[1].replace('.
                        txt', '')})], ignore_index=True)
76              except:
77                  questions, ground_truth, y_hat =
                        padronizar_tamanho_listas(questions,
                        ground_truth, y_hat)
78                  df = pd.concat([df, pd.DataFrame({'question':
                        questions, 'y': ground_truth, 'y_hat': y_hat,
                         'task_number': task.split('_')[1].replace('.
                        txt', '')})], ignore_index=True)
79              sleep(60)
80
81      df.to_csv(f'babi-tasks/results/{test}.csv', index=False)
```

Código-fonte 15 – Arquivo de Testes Para RocStories

22 TESTE PARA ROCSTORIES

```
from CTM import CTM
import warnings
warnings.filterwarnings('ignore')
import os
import random
import pandas as pd
os.environ["TOKENIZERS_PARALLELISM"] = "false"
import re
from time import sleep

def select_random(lista, quantidade=5):
    if len(lista) < quantidade:
        raise ValueError("A lista deve conter pelo menos {} elementos.".format(quantidade))
    return random.sample(lista, quantidade)

def extract_answers(conteudo):
    padrao = r"### Respostas Completas:\s*(\d+|\[.*?\])"
    match = re.search(padrao, conteudo, re.DOTALL)

    if match:
        respostas = match.group(1).strip()# Converte a string da lista para uma lista real
        return respostas
    return None

def padronizar_tamanho_listas(*listas):
    max_length = max(len(lista) for lista in listas)
    return [lista + [None] * (max_length - len(lista)) for lista in listas]

count = 0
df = pd.read_csv(<ROCSTORIES_PATH>)
```

```
tasks = df.sample(25)
output_df = pd.DataFrame(columns=['InputStoryid', 'context', '
    question', 'y', 'y_hat'])

print(f'\nExecutando Tarefas de RocStories')

for i, row in tasks.iterrows():
    context = ' '.join([row['InputSentence1'], row['
        InputSentence2'], row['InputSentence3'], row['
        InputSentence4']])
    question = f'Which of these fragments concludes the story in
         a coherent way? \n1. {row["RandomFifthSentenceQuiz1"]} \
        n2. {row["RandomFifthSentenceQuiz2"]}'
    count += 5


    file1 = "output_report.txt"
    file2 = "uptree_competition.txt"
    files_to_check = [file1, file2]
    for file in files_to_check:
        if os.path.exists(file):
            os.remove(file)
    print(f"\t- Executando Story {row['InputStoryid']}")

    story = [context, question]

    ctm = CTM(8, story)
    ctm.run()

    with open("/Users/guisalesfer/CTM_implementation/ctm_project
        /classes/output_report.txt", "r", encoding="utf-8") as f:
         conteudo = f.read()
    y_hat = extract_answers(conteudo)
```

```
58      final_row = pd.DataFrame({'InputStoryid':[row['InputStoryid']], 'context':[context], 'question':[question], 'y':[row['AnswerRightEnding']], 'y_hat':[y_hat]})
59
60      output_df = pd.concat([output_df, final_row], ignore_index=True)
61      sleep(10)
62
63      if count <= 100:
64          print(f'\n Results Saved ! | Progress Default: {count}%')
65      else:
66          print(f'\n Results Saved ! | Progress Extra: {count+20}%')
67
68  output_df.to_csv(OUTPUT_ROCSTORIES_PATH, index=False)
```

Código-fonte 16 – Arquivo de Testes Para RocStories