Construíndo bons chatbots

Giseldo Neo

Sumário

Capítulo 1

Chatbots: Definições e Contexto

1.1 Introdução

Definição

Um chatbot é um programa de computador que simula uma conversa humana, geralmente utilizando texto ou áudio. Eles oferecem respostas diretas a perguntas e auxiliam em diversas tarefas, servindo tanto para conversas gerais quanto para ações específicas, como abrir uma conta bancária.

Origem do termo

Embora o chatbot ELIZA [38] seja frequentemente considerado um dos primeiros exemplos de software conversacional, o termo "chatbot" ainda não era utilizado na época de sua criação. O termo "chatterbot", sinônimo de "chatbot", foi popularizado por Michael Mauldin em 1994, ao descrever seu programa JULIA [24]. Publicações acadêmicas, como os anais da *Virtual Worlds and Simulation Conference* de 1998 [10], também ajudaram a consolidar o uso do termo.

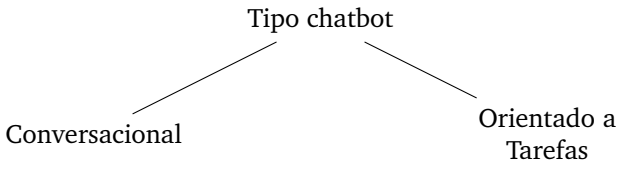
ELIZA

O ELIZA, criado em 1966 por Joseph Weizenbaum, representou um experimento revolucionário na interação humano-computador [38]. Seu script mais famoso, DOCTOR, imitava rudimentarmente um psicoterapeuta, utilizando correspondência de padrões simples. Por exemplo, quando um usuário inseria a frase "Estou triste", o ELIZA respondia "Por que você está triste hoje?", reformulando a entrada como uma pergunta. Seu funcionamento baseia-se em um conjunto de regras que lhe permitem analisar e compreender a linguagem humana de forma limitada e aproximada. Esse tipo de aplicação do ELIZA adequou-se bem a esse domínio, pois dependia de pouco conhecimento sobre o ambiente externo; as regras no script DOCTOR permitiam que o programa respondesse ao usuário com outras perguntas ou simplesmente refletisse a afirmação original. O ELIZA não possuía uma real "compreensão" da linguagem humana; ele apenas utilizava palavras-chave e manipulava frases para que a interação parecesse natural. Uma descrição detalhada do funcionamento do ELIZA, com exemplos em Python, será apresentada em seções posteriores.

ChatGPT

Outro chatbot famoso é o ChatGPT. Desenvolvido pela OpenAI, este é um modelo de linguagem capaz de gerar texto muito semelhante ao criado por humanos. Ele utiliza aprendizagem profunda (deep learning) e redes neurais para gerar sentenças e parágrafos com base nas entradas e informações fornecidas. É capaz de produzir textos coerentes e até mesmo realizar tarefas simples, como responder a perguntas e gerar ideias. Contudo, é importante lembrar que o ChatGPT não possui consciência nem a capacidade de compreender contexto ou emoções. Ele é um exemplo de Modelo de Linguagem Grande (Large Language Model - LLM), baseado na arquitetura conhecida como Transformers, introduzida em 2017 [2]. Esses modelos são treinados com terabytes de texto, utilizando mecanismos de autoatenção que avaliam a relevância de cada palavra em uma frase. Ao contrário das regras manuais do ELIZA,

Figura 1.1: Classificação de tipos de chatbots.



os LLMs extraem padrões linguísticos a partir da vasta quantidade de dados com que a rede neural foi treinada.

Tipo Conversacional

Esses dois chatbots, ELIZA e ChatGPT, são bons representantes de chatbots do tipo conversacional. Os chatbots conversacionais são utilizados para interagir sobre um propósito específico ou mesmo sobre assuntos gerais.

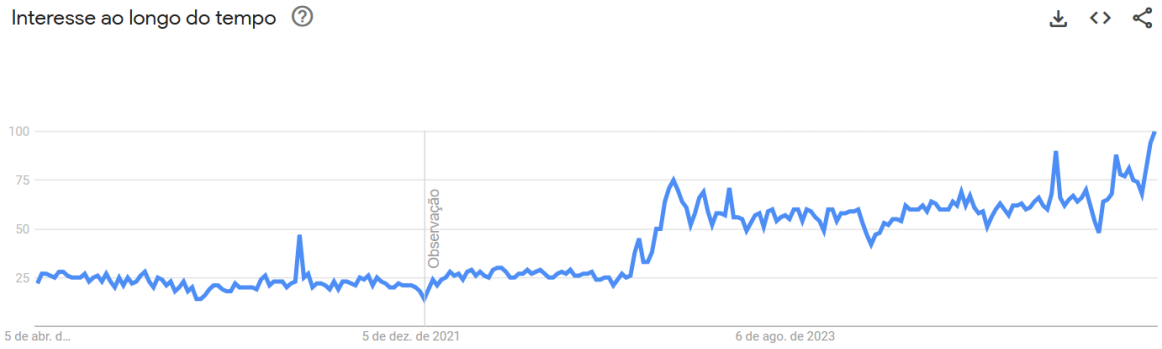
Tipo Orientado a tarefas

Outro tipo de chatbot é o orientado a tarefas. Os chatbots orientados a tarefas executam ações específicas, como abrir uma conta bancária ou pedir uma pizza. Geralmente, as empresas disponibilizam chatbots orientados a tarefas para seus usuários, com regras de negócio embutidas na conversação e com fluxos bem definidos. Normalmente, não se espera pedir uma pizza e, no mesmo chatbot, discutir os estudos sobre Ética do filósofo Immanuel Kant (embora talvez haja quem queira).

Essas duas classificações - conversacional e orientado a tarefas - (Figura 1.1) ainda não são suficientes para uma completa classificação dada diversas caractéristicas e enorme quantidade de chatbots existentes. Existem outras classificações que serão discutidas em seções posteriores.

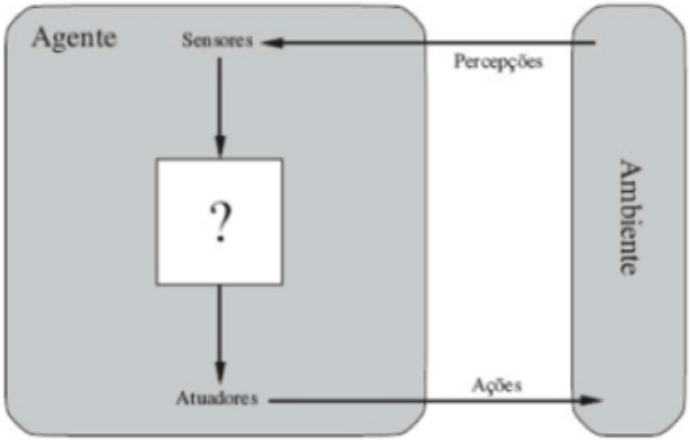
A popularidade dos chatbots tem crescido significativamente em diversos domínios de aplicação [21, 12, 31]. Essa tendência é corroborada pelo aumento do interesse de busca pelo termo “chatbots”, conforme análise de dados do Google Trends no período entre 2020 e 2025 (Figura 1.2). Nesta figura, os valores representam o interesse relativo de busca ao longo do tempo, onde 100 indica o pico de popularidade no período analisado e 0 (ou a ausência de dados) indica interesse mínimo ou dados insuficientes.

Figura 1.2: Evolução do interesse de busca pelo termo “chatbot” (Google Trends, 2020-2025).



Fonte: Google Trends acesso em 05/04/2025

Figura 1.3: Arquitetura conceitual de um agente.



Fonte: Diretamente retirado de [29]

1.2 Chatbots e Agentes: Definições e Distinções

Define-se chatbot como um programa computacional projetado para interagir com usuários por meio de linguagem natural. Por outro lado, o conceito de agente possui uma definição mais ampla: trata-se de uma entidade computacional que percebe seu ambiente por meio de sensores e atua sobre esse ambiente por meio de atuadores [29]. A Figura 1.3 ilustra uma arquitetura conceitual de alto nível para um agente.

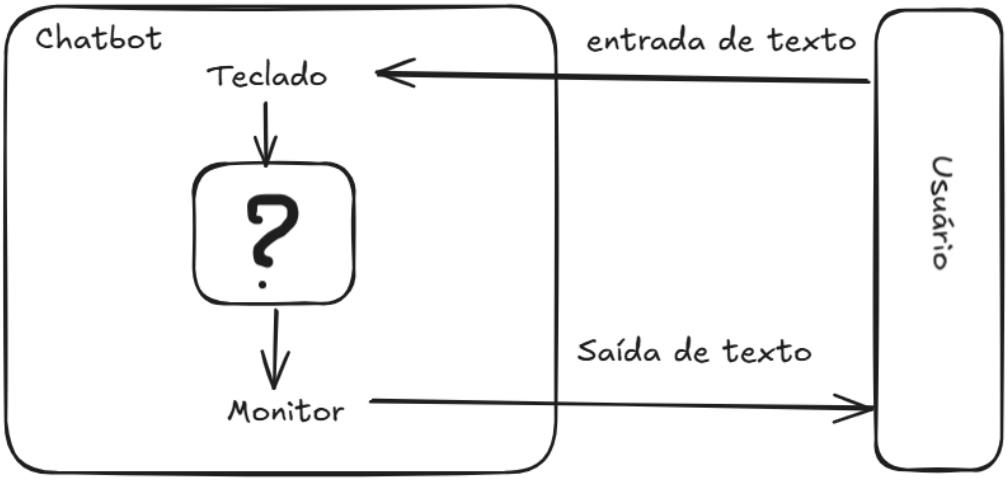
Nesse contexto, um chatbot (Figura 1.4) pode ser considerado uma instanciação específica de um agente, cujo propósito primário é a interação conversacional em linguagem natural.

Com o advento de modelos de linguagem avançados, como os baseados na arquitetura *Generative Pre-trained Transformer* (GPT), observou-se uma recontextualização do termo "agente" no domínio dos sistemas conversacionais. Nessa abordagem mais recente, um sistema focado predominantemente na geração de texto conversacional tende a ser denominado "chatbot". Em contraste, o termo "agente" é frequentemente reservado para sistemas que, além da capacidade conversacional, integram e utilizam ferramentas externas (por exemplo, acesso à internet, execução de código, interação com APIs) para realizar tarefas complexas e interagir proativamente com o ambiente digital. Um sistema capaz de realizar uma compra online, processar um pagamento e confirmar um endereço de entrega por meio do navegador do usuário seria, portanto, classificado como um agente, diferentemente de chatbots mais simples como ELIZA, cujo foco era estritamente o diálogo.

1.3 Gerenciamento do Diálogo e Fluxo Conversacional

A interação textual mediada por chatbots não se constitui em uma mera justaposição aleatória de turnos de conversação ou pares isolados de estímulo-resposta. Pelo contrário, espera-se que a conversação exiba coerência e mantenha relações lógicas e semânticas entre os turnos consecutivos. O

Figura 1.4: Representação esquemática de um chatbot.



Fonte: Giseldo Neo (2025)

estudo da estrutura e organização da conversa humana é abordado por disciplinas como a Análise da Conversação.

Análise da Conversação

No contexto da linguística textual e análise do discurso em língua portuguesa, os trabalhos de Marcuschi [22] são relevantes ao investigar a organização da conversação. Marcuschi analisou a estrutura conversacional em termos de unidades coesas, como o "tópico conversacional", que agrupa turnos relacionados a um mesmo assunto ou propósito interacional.

Fluxo de diálogo

Conceitos oriundos da Análise da Conversação, como a gestão de tópicos, têm sido aplicados no desenvolvimento de chatbots para aprimorar sua capacidade de manter diálogos coerentes e contextualmente relevantes com usuários humanos [25]. Na prática de desenvolvimento de sistemas conversacionais, a estrutura lógica e sequencial da interação é frequentemente modelada e referida como "fluxo de conversação" ou "fluxo de diálogo". Contudo, é importante ressaltar que a implementação explícita de modelos sofisticados de gerenciamento de diálogo, inspirados na Análise da Conversação, não é uma característica universal de todos os chatbots, variando conforme a complexidade e o propósito do sistema. Um exemplo esquemático de um fluxo conversacional é apresentado na Figura 1.5.

Um fluxograma para um chatbot que vende roupas está representada na Figura 1.6.

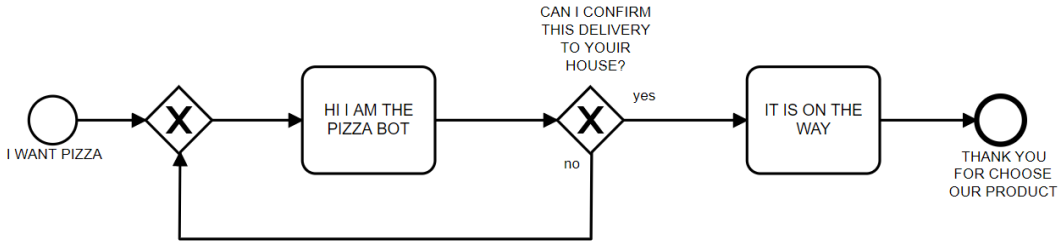
1.4 Depois do ELIZA

ALICE AIML

Um marco significativo na evolução dos chatbots foi o ALICE, que introduziu a Artificial Intelligence Markup Language (AIML), uma linguagem de marcação baseada em XML [36]. A AIML estabeleceu um paradigma para a construção de agentes conversacionais ao empregar algoritmos de correspondência de padrões. Essa abordagem utiliza modelos pré-definidos para mapear as entradas do usuário a respostas correspondentes, permitindo a definição modular de blocos de conhecimento [36].

No contexto brasileiro, um dos primeiros chatbots documentados capaz de interagir em português, inspirado no modelo ELIZA, foi o Cybele [26]. Posteriormente, desenvolveu-se o Elecktra, também

Figura 1.5: Exemplo esquemático de um fluxo conversacional em um chatbot.



Fonte: Giseldo Neo (2025)

em língua portuguesa, com aplicação voltada para a educação a distância [16]. Em um exemplo mais recente de aplicação governamental, no ano de 2019, o processo de inscrição para o Exame Nacional do Ensino Médio (ENEM) foi disponibilizado por meio de uma interface conversacional baseada em chatbot (Figura 1.7).

O desenvolvimento de chatbots avançados tem atraído investimentos de grandes corporações. Notavelmente, a IBM desenvolveu um sistema de resposta a perguntas em domínio aberto utilizando sua plataforma Watson [8]. Esse tipo de tarefa representa um desafio computacional e de inteligência artificial (IA) considerável. Em 2011, o sistema baseado em Watson demonstrou sua capacidade ao competir e vencer competidores humanos no programa de perguntas e respostas JEOPARDY! [8].

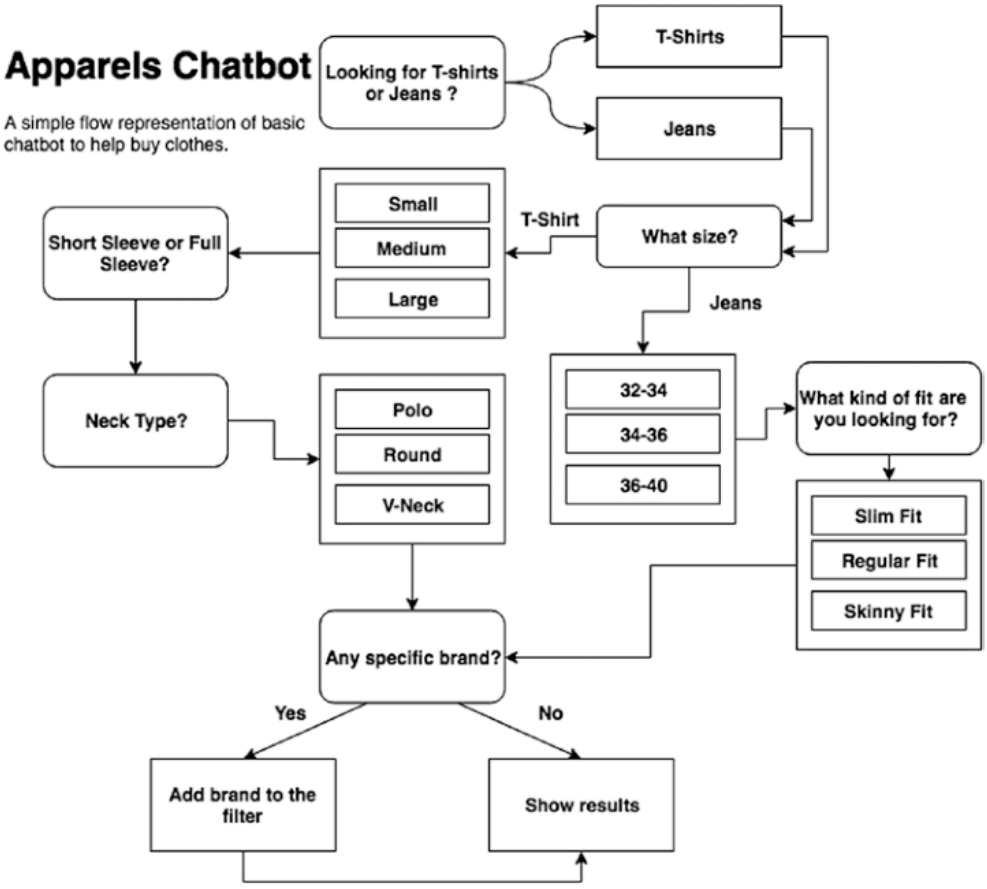
Diversos outros chatbots foram desenvolvidos para atender a demandas específicas em variados domínios. Exemplos incluem: BUTI, um companheiro virtual com computação afetiva para auxiliar na manutenção da saúde cardiovascular [11]; EduBot, um agente conversacional projetado para a criação e desenvolvimento de ontologias com lógica de descrição [18]; PMKLE, um ambiente inteligente de aprendizado focado na educação em gerenciamento de projetos [33]; RENAN, um sistema de diálogo inteligente fundamentado em lógica de descrição [6]; e MOrFEu, voltado para a mediação de atividades cooperativas em ambientes inteligentes na Web [2].

1.5 Abordagens

Desde o pioneirismo do ELIZA, múltiplas abordagens e técnicas foram exploradas para o desenvolvimento de chatbots. Entre as mais relevantes, destacam-se: AIML com correspondência de padrões (pattern matching), análise sintática (Parsing), modelos de cadeia de Markov (Markov Chain Models), uso de ontologias, redes neurais recorrentes (RNNs), redes de memória de longo prazo (LSTMs), modelos neurais sequência-a-sequência (Sequence-to-Sequence), aprendizado adversarial para geração de diálogo, além de abordagens baseadas em recuperação (Retrieval-Based) e generativas (Generative-Based) [3, 28, 30, 1, 17], entre outras.

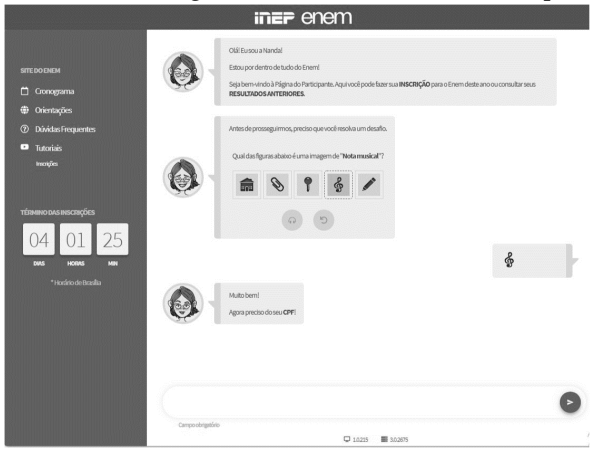
Além disso, diversos frameworks têm sido desenvolvidos para facilitar a criação desses agentes complexos, como CrewAI e bibliotecas associadas a plataformas como Hugging Face (e.g., Transformers Agents), que fornecem abstrações e ferramentas em Python para orquestrar múltiplos componentes e o uso de ferramentas externas.

Figura 1.6: Representação de uma árvore de decisão para comprar roupas online



[27].

Figura 1.7: Interface de chatbot para inscrição no ENEM 2019.



Fonte: Captura de tela realizada por Giseldo Neo.

1.6 Problemática

Apesar do progresso recente de chatbots, como o XiaoIce e o ChatGPT, o mecanismo fundamental da inteligência em nível humano, frequentemente refletido na comunicação, ainda não está totalmente esclarecido [32]. Para avançar na solução desses desafios, serão necessários progressos em diversas áreas da IA cognitiva, tais como: modelagem empática de conversas, modelagem de conhecimento e memória, inteligência de máquina interpretável e controlável, e calibração de recompensas emocionais [32].

Uma das dificuldades na construção de chatbots reside em gerenciar a complexidade das estruturas condicionais ("se-então") que definem o fluxo do diálogo [27]. Quanto maior o número de decisões a serem tomadas, mais numerosas e intrincadas tendem a ser essas estruturas condicionais. Contudo, elas são essenciais para codificar fluxos de conversação complexos. Se a tarefa que o chatbot visa simular é inerentemente complexa e envolve múltiplas condições, o código precisará refletir essa complexidade. Para facilitar a visualização desses fluxos, uma solução eficaz é a utilização de fluxogramas. Embora simples de criar e entender, os fluxogramas constituem uma poderosa ferramenta de representação para este problema.

Os chatbots baseados em AIML (Artificial Intelligence Markup Language) apresentam desvantagens específicas. Por exemplo, o conhecimento é representado como instâncias de arquivos AIML. Se esse conhecimento for criado com base em dados coletados da Internet, ele não será atualizado automaticamente, exigindo atualizações periódicas manuais [20]. No entanto, já existem abordagens para mitigar essa limitação, permitindo carregar conteúdo AIML a partir de fontes como arquivos XML [19], um corpus textual [7] ou dados do Twitter [39].

Outra desvantagem do AIML reside na relativa complexidade de seus padrões de correspondência (patterns). Além disso, a manutenção do sistema pode ser árdua, pois, embora a inserção de conteúdo (categorias) seja conceitualmente simples, grandes volumes de informação frequentemente precisam ser adicionados manualmente [20].

Especificamente no caso do AIML, a construção e a visualização de fluxos de diálogo complexos enfrentam dificuldades adicionais. Devido ao seu formato baseado em texto, muitas vezes é difícil perceber claramente como as diferentes categorias (unidades de conhecimento e resposta) se interligam

para formar a estrutura da conversação.

Capítulo 2

Eliza Explicado

2.1 Introdução

o ELIZA foi um dos primeiros programas de processamento de linguagem natural e foi apresentado em 1966 por Joseph Weizenbaum no MIT [38].

SCRIPT

O conjunto de padrões e respostas predefinidas constitui o que Weizenbaum chamou de "roteiro" (script) de conversa. O mecanismo do ELIZA separa o motor genérico de processamento (o algoritmo de busca de palavras-chave e aplicação de regras) dos dados do script em si. Isso significa que ELIZA podia em teoria simular diferentes personalidades ou tópicos apenas carregando um script diferente, sem alterar o código do programa (veja na Figura 2.1). Também foi codificado um editor de texto para alteração do Script.

Figura 2.1: ELIZA: separação entre o código fonte, o script e o editor de texto.

SCRIPT
DOCTOR

Um destes scripts (o DOCTOR) o deixou famoso. Com este script carregado, o ELIZA simulava um psicoterapeuta (do estilo Rogeriano), refletindo as palavras do usuário de volta em forma de pergunta. A primeira linha do script é a introdução do chatbot, a primeira palavra que ele escrever para o usuário. START é um comando técnico do script, serve para indicar que a seguir virão as regras. REMEMBER 5 estabelece que a prioridade da palavra REMEMBER é 5, ou seja, se o usuário disser algo que contenha a palavra REMEMBER, o ELIZA irá responder com uma das perguntas definidas entre parênteses. A seguir estão as regras de transformação de frases (decomposition rule e reassembly rule) associadas à palavra-chave REMEMBER. O exemplo abaixo mostra como o script DOCTOR é estruturado:

```
(REMEMBER 5
 (
  (0 YOU REMEMBER 0)
  (DO YOU OFTEN THINK OF 4)
```

```
  (DOES THINKING OF ~ BRING ANYTHING ELSE TO MINO)
  (WHAT ELSE OO YOU REMEMBER)
  (WHY DO YOU REMEMBER 4 JUST NOW)
  (WHAT IN THE PRESENT SITUATION REMINDS YOU OF 4)
  (WHAT IS THE CONNECTION BETWEEN ME AND 4)
 )
 (
  (0 DO I REMEMBER 0)
  (DID YOU THINK I WOULD FORGET 5)
  (WHY DO YOU THINK I SHOULD RECALL 5 NOW)
  (WHAT ABOUT 5)
  (=WHAT)
  (YOU MENTIONED S)
 )
)

(IF 3
 (
  (0 IF 0)
  (DO YOU THINK ITS LIKELY THAT 3)
  (DO YOU WISH THAT 3)
  (WHAT DO YOU THINK ABOUT 3)
  (REALLY, 2 3)
 )
)
```

A seguir é apresentado um exemplo de diálogo (retirado do artigo original) entre um usuário humano e ELIZA, mostrando as respostas em letras maiúsculas. Nesse exemplo, o usuário faz afirmações pessoais (“Men are all alike.”, “Well, my boyfriend made me come here.”) e ELIZA responde com perguntas ou reformulações (“IN WHAT WAY”, “YOUR BOYFRIEND MADE YOU COME HERE?”), reutilizando termos do próprio usuário. Esse estilo reflexivo encoraja o usuário a continuar falando sem fornecer respostas diretas.

```
Men are all alike.
IN WHAT WAY
They’re always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE
Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE
He says I’m depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
It’s true. I am unhappy.
DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
I need some help, that much seems certain.
WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
```

```
Perhaps I could learn to get along with my mother.
TELL ME MORE ABOUT YOUR FAMILY
My mother takes care of me.
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU
My father.
YOUR FATHER
You are like my father in some ways.
WHAT RESEMBLANCE DO YOU SEE
You are not very aggressive but I think you don't want me to
notice that.
WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE
You don't argue with me.
WHY DO YOU THINK I DON'T ARGUE WITH YOU
You are afraid of me.
DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU
My father is afraid of everybody.
WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER
Bullies.
DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR BOYFRIEND MADE YOU COME HERE
```

2.2 Processamento de Entradas e Palavras-Chave

Pattern Matching

O funcionamento do ELIZA baseia-se em correspondência de padrões por palavras-chave (técnica também chamado casamento de padrão, em inglês *pattner matching* ou baseado em regras *rule-based*). A cada turno, o texto de entrada do usuário é analisado em busca de palavras-chave predefinidas. O ELIZA fazia uma varredura da sentença da esquerda para a direita, identificando se alguma palavra correspondia a uma palavra-chave do script. Caso encontrasse, ELIZA selecionava a palavra-chave mais "importante" (havia um peso de prioridade associado a cada palavra-chave) e ignorava o restante da entrada.

Por exemplo, o script DOCTOR, definia palavras-chave como "ALIKE" ou "SAME" com alta prioridade; assim, na frase "Men are all alike." o programa detectava a palavra "ALIKE" e disparava uma resposta associada a ela (no caso: "In what way?"). Se múltiplas palavras-chave aparecessem, ELIZA escolhia aquela de maior peso para formular a resposta.

Primeiro o texto de entrada digitado pelo usuário era separado em palavras, em um técnica que hoje chamamos de tokenização de palavras. A palavra-chave era identificada, comparando-a sequencialmente até o fim das palavras existentes, ou até ser encontrado uma pontuação. Caso fosse encontrado uma pontuação (ponto final ou vírgula), o texto após a pontuação era ignorado se já tivesse sido identificado uma palavra-chave. Assim cada processamento da resposta foca em apenas uma única afirmação (ou frase) do usuário. Se várias palavras-chave fossem encontradas antes da pontuação, a de maior peso era selecionada.

Por exemplo, o usuário entra com o texto: "I am sick. but, today is raining". Se houvesse uma palavra-chave no script rankeando a palavra "SICK"com alta prioridade, a entrada processada seria somente "I am sick", o restante depois da pontuação (neste caso, o ponto) seria gnorado pelo programa.

Se nenhuma palavra-chave fosse encontrada na entrada, ELIZA recorria a frases genéricas programadas, chamadas de respostas vazias ou sem conteúdo. Nesses casos, o chatbot emitia mensagens do

tipo “I see.” ou “Please, go on.” . Esse mecanismo evitava silêncio quando o usuário dizia algo fora do escopo do script.

Além disso, a implementação original incluía uma estrutura de memória: algumas declarações recentes do usuário eram armazenadas e, se uma entrada subsequente não contivesse novas keywords, ELIZA poderia recuperar um tópico anterior e introduzi-lo na conversa. Por exemplo, se o usuário mencionasse família em um momento e depois fizesse uma afirmação vaga, o programa poderia responder retomando o assunto da família (“DOES THAT HAVE ANYTHING TO DO WITH YOUR FAMILY?”). Essa estratégia dava uma pseudo-continuidade ao diálogo, simulando que o sistema “lembrava” de informações fornecidas anteriormente.

2.3 Regras de Transformação de Frases

Encontrada a palavra-chave, ELIZA aplicava uma regra de transformação associada a ela para gerar a resposta. As regras são definidas em pares: um padrão de análise (decomposition rule) e um modelo de reconstrução de frase (reassembly rule).

Primeiro, a frase do usuário é decomposta conforme um padrão que identifica a contexto mínimo em torno da palavra-chave. Essa decomposição frequentemente envolve separar a frase em partes e reconhecer pronomes ou estruturas gramaticais relevantes. Por exemplo, considere a entrada “You are very helpful.”. Uma regra de decomposição pode identificar a estrutura “You are X” — onde “X” representa o restante da frase — e extrair o complemento “very helpful” como um componente separado.

Em seguida, a regra de reassembly correspondente é aplicada, remontando uma sentença de resposta em que “X” é inserido em um template pré-definido. No exemplo dado, o template de resposta poderia ser “What makes you think I am X?”; ao inserir X = “very helpful”, gera-se “What makes you think I am very helpful?”. Observe que há uma inversão de pessoa: o pronome “you” do usuário foi trocado por “I” na resposta do bot.

De fato, uma parte importante das transformações do ELIZA envolve substituir pronomes (eu/você, meu/seu) para que a resposta faça sentido como uma frase do ponto de vista do computador falando com o usuário. Esse algoritmo de substituição é relativamente simples (por exemplo, “meu” $\rightarrow$ “seu”, “eu” $\rightarrow$ “você”, etc.), mas essencial para dar a impressão de entendimento gramatical.

2.4 Implementação Original e Variações Modernas

A implementação original de ELIZA foi feita em uma linguagem chamada MAD-SLIP (um dialecto de Lisp) rodando em um mainframe IBM 7094 no sistema CTSS do MIT. O código fonte do programa principal continha o mecanismo de correspondência, enquanto as regras de conversação (script DOCTOR) eram fornecidas separadamente em formato de listas associativas, similar a uma lista em Lisp. Infelizmente, Weizenbaum não publicou o código completo no artigo de 1966 (o que era comum na época), mas décadas depois o código em MAD-SLIP foi recuperado nos arquivos do MIT, comprovando os detalhes de implementação [15]. De qualquer forma, a arquitetura descrita no artigo influenciou inúmeras reimplementações acadêmicas e didáticas nos anos seguintes.

Diversos entusiastas e pesquisadores reescreveram ELIZA em outras linguagens de programação, dada a simplicidade relativa de seu algoritmo. Ao longo dos anos surgiram versões em Lisp, PL/I, BASIC, Pascal, Prolog, Java, Python, OZ, JavaScript, entre muitas outras. Cada versão normalmente incluía o mesmo conjunto de regras do script terapeuta ou pequenas variações.

As ideias de ELIZA também inspiraram chatbots mais avançados. Poucos anos depois, em 1972, surgiu PARRY, escrito pelo psiquiatra Kenneth Colby, que simulava um paciente paranoico. PARRY tinha um modelo interno de estado emocional e atitudes, mas na camada de linguagem ainda usava muitas respostas baseadas em regras, chegando a "conversar" com o próprio ELIZA em experimentos da época.

Em 1995, Richard Wallace desenvolveu o chatbot ALICE (Artificial Linguistic Internet Computer Entity), que levava o paradigma de ELIZA a uma escala muito maior. ALICE utilizava um formato XML chamado AIML (Artificial Intelligence Markup Language) para definir milhares de categorias de padrões e respostas. Com mais de 16.000 templates mapeando entradas para saídas [36], ALICE conseguia manter diálogos bem mais naturais e abrangentes que o ELIZA original, embora o princípio básico de correspondência de padrões permanecesse. Esse avanço rendeu a ALICE três vitórias no Prêmio Loebner (competição de chatbots) no início dos anos 2000 [36].

Outras variações e sucessores notáveis incluem Jabberwacky (1988) – que já aprendia novas frases – e uma profusão de assistentes virtuais e bots de domínio específico nas décadas seguintes [36]. Em suma, o legado de ELIZA perdurou por meio de inúmeros chatbots baseados em regras, até a transição para abordagens estatísticas e de aprendizado de máquina no final do século XX.

2.5 Comparação com Modelos de Linguagem Modernos

A técnica de ELIZA, baseada em palavras-chave com respostas predefinidas, contrasta fortemente com os métodos de Large Language Models (LLMs) atuais, como o GPT-4, que utilizam redes neurais de milhões (ou trilhões) de parâmetros e mecanismos de atenção.

2.5.1 Mecanismo de Pesos: Palavras-Chave vs. Atenção Neural

No ELIZA, a "importância" de uma palavra era determinada manualmente pelo programador através de pesos ou rankings atribuídos a certas palavras-chave no script. Ou seja, o programa não aprendia quais termos focar – ele seguia uma lista fixa de gatilhos. Por exemplo, termos como "sempre" ou "igual" tinham prioridade alta no script DOCTOR para garantir respostas apropriadas.

Em contraste, modelos modernos como o GPT não possuem uma lista fixa de palavras importantes; em vez disso, eles utilizam o mecanismo de self-attention para calcular dinamicamente pesos entre todas as palavras da entrada conforme o contexto [34].

Na arquitetura Transformer, cada palavra (token) de entrada gera consultas e chaves que interagem com todas as outras, permitindo ao modelo atribuir pesos maiores às palavras mais relevantes daquela frase ou parágrafo [34]. Em outras palavras, o modelo aprende sozinho quais termos ou sequências devem receber mais atenção para produzir a próxima palavra na resposta. Esse mecanismo de atenção captura dependências de longo alcance e nuances contextuais que um sistema de palavras-chave fixas como o ELIZA não consegue representar.

Além disso, o "vocabulário" efetivo de um LLM é imenso – um moddelo GPT pode ser treinado com trilhões de palavras e ter ajustado seus parâmetros para modelar estatisticamente a linguagem humana [34]. Como resultado, pode-se dizer metaforicamente que os LLMs têm uma lista de "palavras-chave" milhões de vezes maior (na prática, distribuída em vetores contínuos) e um método bem mais sofisticado de calcular respostas do que o ELIZA.

Enquanto ELIZA dependia de coincidências exatas de termos para disparar regras, modelos como

GPT avaliam similaridades semânticas e contexto histórico graças às representações densas (embeddings) aprendidas durante o treinamento de rede neural.

2.5.2 Contextualização e Geração de Linguagem

Devido à sua abordagem baseada em regras locais, o ELIZA tinha capacidade de contextualização muito limitada. Cada input do usuário era tratado quase isoladamente: o programa não construía uma representação acumulada da conversa, além de artifícios simples como repetir algo mencionado (a estrutura de memória) ou usar pronomes para manter a ilusão de continuidade. Se o usuário mudasse de tópico abruptamente, o ELIZA não "perceberia" – ele apenas buscaria a próxima palavra-chave disponível ou recorreria a frases genéricas.

Em contraste, modelos de linguagem modernos levam em conta um longo histórico de diálogo. Chatbots que usam GPT podem manter um contexto centenas ou milhares de tokens (palavras ou fragmentos) em sua janela de atenção, o que significa que eles conseguem referenciar informações mencionadas vários parágrafos atrás e integrá-las na resposta corrente. O mecanismo de self-attention, em particular, permite que o modelo incorpore relações contextuais complexas: cada palavra gerada pode considerar influências de palavras distantes no texto de entrada [34].

Por exemplo, ao conversar com um LLM, se você mencionar no início da conversa que tem um irmão chamado Alex e depois perguntar "ele pode me ajudar com o problema?", o modelo entenderá que "ele" se refere ao Alex mencionado anteriormente (desde que dentro da janela de contexto). Já o ELIZA original não teria como fazer essa ligação, a menos que houvesse uma regra explícita para "ele" e algum armazenamento específico do nome – algo impraticável de antecipar via regras fixas para todos os casos.

Outra diferença crucial está na geração de linguagem. O ELIZA não gera texto original no sentido pleno: suas respostas são em grande parte frases prontas (ou templates fixos) embaralhadas com partes da fala do usuário. Assim, seu vocabulário e estilo são limitados pelo script escrito manualmente. Modelos GPT, por sua vez, geram respostas novas combinando probabilisticamente o conhecimento adquirido de um extenso corpus. Eles não se restringem a repetir trechos da entrada, podendo elaborar explicações, fazer analogias, criar perguntas inéditas – tudo coerente com os exemplos linguísticos em sua base de treinamento. Enquanto ELIZA tendia a responder com perguntas genéricas ou devolvendo as palavras do usuário, os LLMs podem produzir respostas informativas e detalhadas sobre o assunto (pois "aprenderam" uma ampla gama de tópicos durante o treinamento). Por exemplo, se perguntarmos algo factual ou complexo, o ELIZA falharia por não ter nenhuma regra a respeito, provavelmente dando uma resposta vazia. Já um modelo como GPT-4 tentará formular uma resposta baseada em padrões linguísticos aprendidos e em conhecimento implícito dos dados, muitas vezes fornecendo detalhes relevantes.

Em termos de fluência e variedade, os modelos modernos superam o ELIZA amplamente. O ELIZA frequentemente se repetia ou caía em loops verbais quando confrontado com inputs fora do roteiro – um limite claro de sistemas por regras estáticas. Os LLMs produzem linguagem muito mais natural e adaptável, a ponto de muitas vezes enganarem os usuários sobre estarem conversando com uma máquina (um efeito buscado desde o Teste de Turing). Ironicamente, ELIZA nos anos 60 já provocou um precursor desse fenômeno – o chamado Efeito ELIZA, em que pessoas atribuem compreensão ou sentimentos a respostas de computador que, na verdade, são superficiais. Hoje, em chatbots GPT, esse efeito se intensifica pela qualidade das respostas, mas a distinção fundamental permanece: ELIZA seguia scripts sem compreender, enquanto LLMs inferem padrões e significados de forma estatística, sem entendimento consciente, mas atingindo resultados que simulam compreensão de maneira muito

mais convincente.

Em resumo, os avanços de arquitetura (especialmente o mecanismo de atenção) ampliaram drasticamente a capacidade de contextualização e geração dos chatbots modernos, marcando uma evolução significativa desde o mecanismo simples porém pioneiro de ELIZA.

Capítulo 3

Artificial Intelligence Markup Language (AIML)

O Artificial Intelligence Markup Language (AIML) é uma especificação baseada em XML, proposta por [37], destinada à programação de agentes conversacionais, comumente denominados chatbots. A concepção da linguagem prioriza o minimalismo, característica que simplifica o processo de criação de bases de conhecimento por indivíduos sem experiência prévia em programação [37]. A arquitetura fundamental de um interpretador AIML genérico é ilustrada na Figura 3.1.

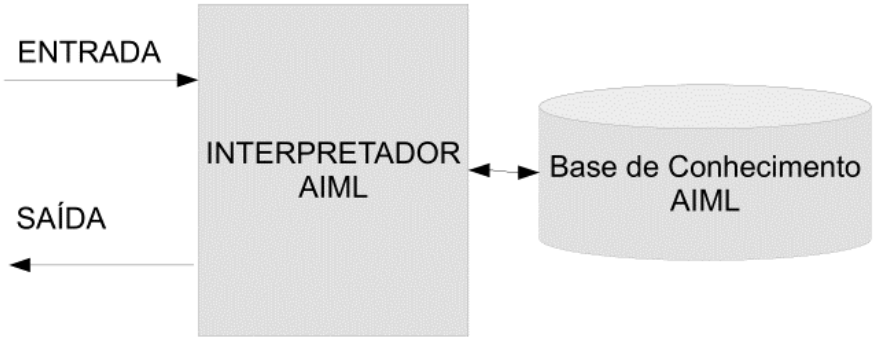
A técnica central empregada pelo AIML é a correspondência de padrões (*pattern matching*). Este método é amplamente utilizado no desenvolvimento de chatbots, particularmente em sistemas orientados a perguntas e respostas [1]. Uma das metas de projeto do AIML é possibilitar a fusão de bases de conhecimento de múltiplos chatbots especializados em domínios distintos. Teoricamente, um interpretador poderia agregar essas bases, eliminando automaticamente categorias redundantes para formar um *chatbot* mais abrangente [36].

AIML é frequentemente associado aos chatbots de terceira geração [23] e estima-se sua adoção em mais de 50.000 implementações em diversos idiomas. Extensões da linguagem foram propostas, como o iAIML, que introduziu novas *tags* e incorporou o conceito de intenção com base nos princípios da Teoria da Análise da Conversação (TAC) [25]. Adicionalmente, ferramentas baseadas na Web foram desenvolvidas para apoiar a construção de bases de conhecimento AIML [13]. Um exemplo proeminente é o *chatbot* ALICE, cuja implementação em AIML compreendia aproximadamente 16.000 categorias, cada uma potencialmente contendo múltiplas *tags* XML aninhadas [36]. Uma representação visual desta estrutura de conhecimento é apresentada na Figura 3.2.

[36] estabeleceu analogias entre o funcionamento de interpretadores AIML e a teoria do Raciocínio Baseado em Casos (RBC). Nessa perspectiva, as categorias AIML funcionam como "casos", onde o algoritmo identifica o padrão que melhor se alinha à entrada do usuário. Cada categoria estabelece um vínculo direto entre um padrão de estímulo e um modelo de resposta. Consequentemente, chatbots AIML inserem-se na tradição da robótica minimalista, reativa ou de estímulo-resposta [36], conforme esquematizado na Figura 3.3. Vale notar que a própria técnica de RBC já foi integrada a interpretadores AIML como um mecanismo para consultar fontes de dados externas e expandir a base de conhecimento do agente [14].

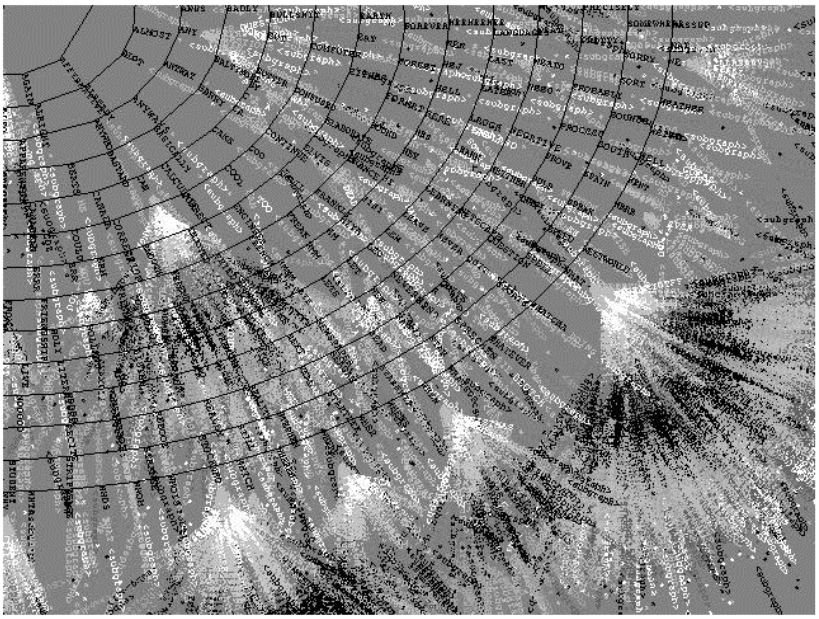
chatbots que utilizam AIML são classificados como sistemas "baseados em recuperação"(*retrieval-based*). Tais modelos operam a partir de um repositório de respostas predefinidas, selecionando a

Figura 3.1: Interpretador AIML arquitetura.



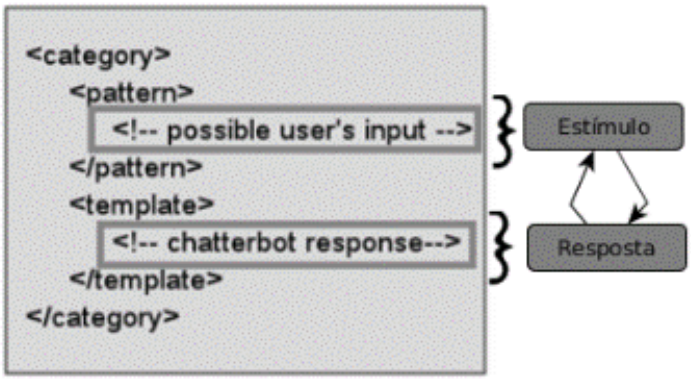
Adaptado de [5]

Figura 3.2: Representação visual da base de conhecimento do chatbot ALICE.



[35]

Figura 3.3: Teoria estímulo-resposta aplicada no AIML



[18]

Figura 3.4: Exemplo de uma base de conhecimento em AIML

```
<aiml>
<category>
 <pattern>*</pattern>
 <template>Hello!</template>
</category>
</aiml
```

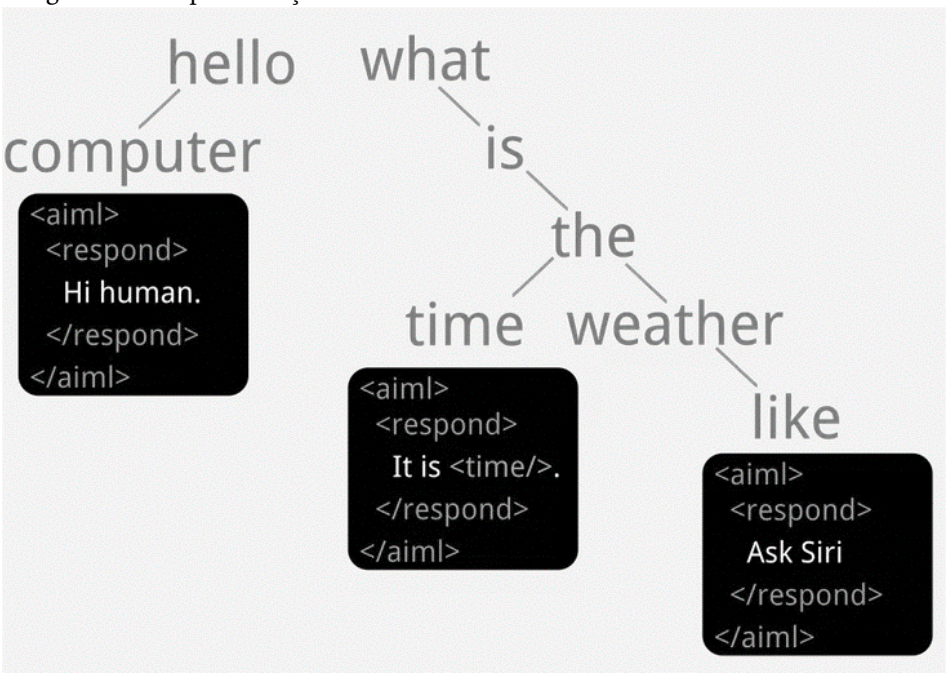
[36]

mais apropriada com base na entrada do usuário e no contexto conversacional, guiando assim o fluxo da interação. Esta abordagem é frequentemente empregada na construção de chatbots destinados a operar em domínios de conhecimento restritos [3].

A Figura 3.4 demonstra a estrutura elementar de um arquivo AIML. A *tag* `<category>` encapsula a unidade básica de conhecimento. Internamente, a *tag* `<pattern>` define o padrão de entrada a ser reconhecido (no exemplo, o caractere curinga `*`, que corresponde a qualquer entrada), enquanto a *tag* `<template>` contém a resposta associada. No exemplo ilustrado, o *chatbot* responderia "Hello!"a qualquer interação. Uma visão abstrata da árvore de conhecimento resultante pode ser observada na Figura 3.5. O AIML padrão suporta transições baseadas primariamente em correspondência de padrões, uma limitação inerente, embora extensões específicas de interpretadores possam permitir a integração de outras técnicas de processamento.

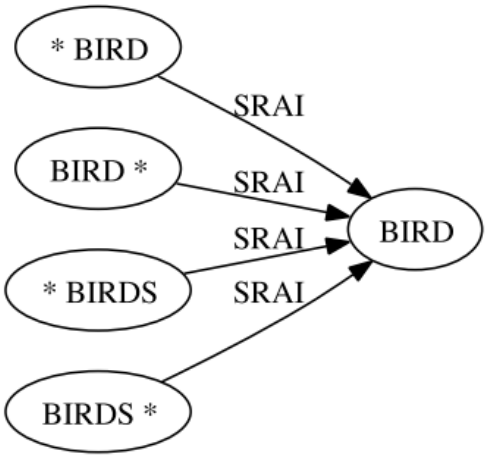
O profissional responsável pela criação, manutenção e curadoria da base de conhecimento de um *chatbot* AIML é denominado *botmaster* [36]. Suas atribuições englobam a edição da base (frequentemente via ferramentas auxiliares), a análise de logs de diálogo para identificar padrões de interação e a subsequente criação ou refino de respostas. Este papel pode ser exercido por indivíduos com diferentes perfis, incluindo *webmasters*, desenvolvedores, redatores, engenheiros ou outros interessados

Figura 3.5: Representação visual abstrata de uma base de conhecimento AIML



https://www.pandorabots.com/docs/aiml-fundamentals/

Figura 3.6: Uso da tag <srai>



[7]

na construção de chatbots [36].

Algumas implementações de interpretadores AIML podem incorporar capacidades rudimentares de compreensão semântica através do *Resource Description Framework* (RDF)[1]. O RDF é um padrão W3C para representação de informações na Web, usualmente por meio de triplas (sujeito-predicado-objeto) que descrevem relações entre entidades. No contexto AIML, RDF pode ser utilizado para armazenar e consultar fatos. Contudo, mesmo com tais adições, as capacidades linguísticas permanecem aquém da complexidade e do potencial gerativo da linguagem humana, conforme descrito por [4].

Embora [9] argumente que o AIML padrão carece de um conceito explícito de "intenção"(*intent*), similar ao encontrado em plataformas de *Natural Language Understanding* (NLU), é possível emular o reconhecimento de intenções. Isso é tipicamente alcançado definindo categorias que representam "formas canônicas"ou "padrões atômicos"para uma intenção específica[2]. Variações de entrada (e.g., "oi", "olá") podem ser mapeadas para uma categoria canônica (e.g., "saudação") usando a *tag* <srai> (*Symbolic Reduction Artificial Intelligence*), que redireciona o fluxo de processamento (ver Figura 3.6). Dessa forma, um *chatbot* AIML pode gerenciar intenções distintas dentro de seu domínio, como realizar um pedido ou verificar o status de entrega.

chatbots baseados em AIML têm obtido sucesso significativo em competições como o Prêmio Loebner. Notavelmente, o *chatbot* Mitsuku[3], desenvolvido por Steve Worswick, conquistou múltiplos títulos recentes[4], seguindo vitórias anteriores do ALICE. [36].

Adicionalmente, Mitsuku foi classificado em primeiro lugar numa análise comparativa envolvendo oito chatbots [31]. Nesse estudo, que avaliou atributos conversacionais com base em um conjunto

[1] https://github.com/keiffster/program-y/wiki/RDF
[2] https://medium.com/pandorabots-blog/new-feature-visualize-your-aiml-26e33a590da1
[3] https://www.pandorabots.com/mitsuku/
[4] https://aisb.org.uk/category/loebner-prize/

padronizado de perguntas, o Google Assistant obteve a segunda posição, seguido pela Siri em terceiro. O *chatbot* ALICE. alcançou a quarta posição, enquanto o ELIZA ficou na última colocação entre os sistemas comparados [31].

3.1 Tags do AIML 1.0: Explicação e Exemplos

Esta seção descreve as principais tags do AIML, versão 1.0, com explicações e exemplos.

`<aiml>` **Descrição:** Tag raiz que engloba todo o conteúdo AIML.
Exemplo:

```
<aiml version="1.0">
 <!-- Categorias aqui -->
</aiml>
```

`<category>` **Descrição:** Unidade básica de conhecimento, contendo um padrão e uma resposta.
Exemplo:

```
<category>
 <pattern>OLÁ</pattern>
 <template>Oi! Como posso ajudar você hoje?</template>
</category>
```

`<pattern>` **Descrição:** Define o padrão de entrada do usuário, com curingas como `*` e `_`.
Exemplo:

```
<category>
 <pattern>EU GOSTO DE *</pattern>
 <template>Que bom que você gosta de <star/>!</template>
</category>
```

`<template>` **Descrição:** Define a resposta do bot ao padrão correspondente.
Exemplo:

```
<category>
 <pattern>QUAL É O SEU NOME</pattern>
 <template>Meu nome é Grok.</template>
</category>
```

`<star/>` **Descrição:** Captura o conteúdo do curinga `*` ou `_`.
Exemplo:

```
<category>
 <pattern>MEU NOME É *</pattern>
 <template>Olá, <star/>!</template>
</category>
```

`<that>` **Descrição:** Considera a última resposta do bot para decidir a próxima.
Exemplo:

```
<category>
 <pattern>SIM</pattern>
 <that>Você gosta de programar?</that>
 <template>Ótimo! Qual linguagem você prefere?</template>
</category>
```

`<topic>` **Descrição:** Define um contexto ou tópico para categorias.
Exemplo:

```
<category>
  <pattern>VAMOS FALAR SOBRE ESPORTE</pattern>
  <template>Ok! <topic name="esporte"/></template>
</category>
```

`<random>` e `<li>` **Descrição:** Escolhe aleatoriamente uma resposta de uma lista.
Exemplo:

```
<category>
  <pattern>COMO ESTÁ O TEMPO</pattern>
  <template>
    <random>
      <li>Está ensolarado!</li>
      <li>Está chovendo.</li>
    </random>
  </template>
</category>
```

`<condition>` **Descrição:** Adiciona lógica condicional baseada em variáveis.
Exemplo:

```
<category>
  <pattern>COMO EU ESTOU</pattern>
  <template>
    <condition name="humor">
      <li value="feliz">Você está bem!</li>
      <li>Não sei ainda!</li>
    </condition>
  </template>
</category>
```

`<set>` e `<get>` **Descrição:** Define e recupera variáveis.
Exemplo:

```
<category>
 <pattern>MEU NOME É *</pattern>
 <template><set name="nome"><star/></set>Olá, <get name="nome"/>!</template>
</category>
```

`<srai>` **Descrição:** Redireciona a entrada para outro padrão.
Exemplo:

```
<category>
 <pattern>OI</pattern>
 <template><srai>OLÁ</srai></template>
</category>
```

`<think>` **Descrição:** Executa ações sem exibir o conteúdo.
Exemplo:

```
<category>
 <pattern>EU SOU TRISTE</pattern>
 <template><think><set name="humor">triste</set></think>Sinto muito!</template>
</category>
```

`<person>`, `<person2>`, `<gender>` **Descrição:** Transforma pronomes ou ajusta gênero.
Exemplo:

```
<category>
 <pattern>EU TE AMO</pattern>
 <template><person><star/></person> ama você também!</template>
</category>
```

`<formal>`, `<uppercase>`, `<lowercase>` **Descrição:** Formata texto (capitaliza, maiúsculas, minúsculas).
Exemplo:

```
<category>
  <pattern>MEU NOME É joão</pattern>
  <template>Olá, <formal><star/></formal>!</template>
</category>
```

`<sentence>` **Descrição:** Formata como frase (primeira letra maiúscula, ponto final).
Exemplo:

```
<category>
  <pattern>oi</pattern>
  <template><sentence><star/></sentence></template>
</category>
```

Capítulo 4

Chatbot ELIZA em Python

Apresenta-se, nesta seção, uma implementação simplificada em Python de um chatbot inspirado no paradigma ELIZA. Esta implementação demonstra a utilização de expressões regulares para a identificação de padrões textuais (palavras-chave) na entrada fornecida pelo usuário e a subsequente geração de respostas, fundamentada em regras de transformação predefinidas manualmente.


```
import re
import random

regras = [
    (re.compile(r'\b(hello|hi|hey)\b', re.IGNORECASE),
     ["Hello. How do you do. Please tell me your problem."]),

    (re.compile(r'\b(I am|I\'?m) (.+)', re.IGNORECASE),
     ["How long have you been {1}?",
      "Why do you think you are {1}?"]),

    (re.compile(r'\bI need (.+)', re.IGNORECASE),
     ["Why do you need {1}?",
      "Would it really help you to get {1}?"]),

    (re.compile(r'\bI can\'?t (.+)', re.IGNORECASE),
     ["What makes you think you can't {1}?",
      "Have you tried {1}?"]),

    (re.compile(r'\bmy (mother|father|mom|dad)\b', re.IGNORECASE),
     ["Tell me more about your family.",
      "How do you feel about your parents?"]),

    (re.compile(r'\b(sorry)\b', re.IGNORECASE),
```

```
     ["Please don't apologize."]),

    (re.compile(r'\b(maybe|perhaps)\b', re.IGNORECASE),
     ["You don't seem certain."]),

    (re.compile(r'\bbecause\b', re.IGNORECASE),
     ["Is that the real reason?"]),

    (re.compile(r'\b(are you|do you) (.+)\?$', re.IGNORECASE),
     ["Why do you ask that?"]),

    (re.compile(r'\bcomputer\b', re.IGNORECASE),
     ["Do computers worry you?"]),
]

respostas_padrao = [
    "I see.",
    "Please tell me more.",
    "Can you elaborate on that?"
]

def response(entrada_usuario):
    for padrao, respostas in regras:
        match = padrao.search(entrada_usuario)
        if match:
            resposta = random.choice(respostas)
            if match.groups():
                resposta = resposta.format(*match.groups())
            return resposta
    return random.choice(respostas_padrao)


# Exemplo de uso
print("User: Hello.")
print("Bot: " + response("Hello."))

print("User: I am feeling sad.")
print("Bot: " + response("I am feeling sad."))

print("Maybe I was not good enough.")
print("Bot: " + response("Maybe I was not good enough."))

print("My mother tried to help.")
print("Bot: " + response("My mother tried to help."))
```

Na implementação, são definidos múltiplos padrões de expressões regulares que correspondem

a palavras-chave ou estruturas frasais de interesse (e.g., saudações, construções como "I am" ou "I need", referências a termos familiares). A função `response`, ao receber uma string de entrada, itera sequencialmente sobre essas regras. Para cada regra, utiliza-se o método `padrao.search(entrada)` para verificar a ocorrência do padrão correspondente na sentença do usuário.

Quando uma correspondência (match) é encontrada, uma resposta associada à regra é selecionada. É relevante notar que diversas respostas predefinidas contêm marcadores (placeholders), como `{1}`. Estes marcadores são dinamicamente substituídos pelo conteúdo dos grupos capturados pela expressão regular na entrada original do usuário. Por exemplo, se a entrada for *"I am sad"* e esta corresponder a um padrão como `r"I am (.*)"`, o grupo capturado (`(.*)`) conterá a string *"sad"*. Se a resposta associada for *"Why do you think you are {1}?"*, o sistema a formatará como *"Why do you think you are sad?"*, substituindo `{1}` pelo conteúdo do primeiro grupo capturado.

Na ausência de correspondência com qualquer regra específica, o sistema seleciona aleatoriamente uma resposta genérica a partir de um conjunto predefinido (denominado, por exemplo, `respostas_padrao`), tal como *"I see."*.

4.0.1 Exemplo de Interação

```
Usuário: Hello
Bot: Hello. How do you do. Please tell me your problem.

Usuário: I am feeling sad
Bot: Why do you think you are feeling sad?

Usuário: Because I lost my job
Bot: Is that the real reason?

Usuário: Maybe I was not good enough.
Bot: You don't seem certain.

Usuário: My mother tried to help.
Bot: Tell me more about your family.
```

Observa-se na interação que o chatbot inicialmente respondeu à saudação (*"Hello..."*) com base na regra 1. Em seguida, a entrada *"I am sad"* ativou a regra 2, resultando em uma resposta que incorpora o termo *"sad"*. A interação prossegue com o sistema respondendo a entradas iniciadas por *"Because..."* (regra 8) e *"Maybe..."* (regra 7), entre outras. Cada resposta gerada foi determinada pela regra correspondente ativada e pelo conteúdo específico da sentença do usuário, mimetizando o comportamento baseado em padrões do sistema ELIZA original [1].

Capítulo 5

LangChain

LangChain é uma biblioteca de software de código aberto projetada para simplificar a interação com Large Language Models (LLMs) e construir aplicativos de processamento de linguagem natural robustos. Ele fornece uma camada de abstração de alto nível sobre as complexidades de trabalhar diretamente com modelos de linguagem, tornando mais acessível a criação de aplicativos de compreensão e geração de linguagem.

5.1 Por que usar LangChain?

Trabalhar com LLMs pode ser complexo devido à sua natureza sofisticada e aos requisitos de recursos computacionais. LangChain lida com muitos detalhes complexos em segundo plano, permitindo que os desenvolvedores se concentrem na construção de aplicativos de linguagem eficazes. Aqui estão algumas vantagens do uso do LangChain:

- Simplicidade: LangChain oferece uma API simples e intuitiva, ocultando os detalhes complexos de interação com LLMs. Ele abstrai as nuances de carregar modelos, gerenciar recursos computacionais e executar previsões.
- Flexibilidade: A biblioteca suporta vários frameworks de deep learning, como TensorFlow e PyTorch, e pode ser integrada a diferentes LLMs. Isso oferece aos desenvolvedores a flexibilidade de escolher as ferramentas e modelos que melhor atendem às suas necessidades.
- Extensibilidade: LangChain é projetado para ser extensível, permitindo que os usuários criem seus próprios componentes personalizados. Você pode adicionar novos modelos, adaptar o processamento de texto ou desenvolver recursos específicos do domínio para atender aos requisitos exclusivos do seu aplicativo.
- Comunidade e suporte: LangChain tem uma comunidade ativa de desenvolvedores e pesquisadores que contribuem para o projeto. A documentação abrangente, tutoriais e suporte da comunidade tornam mais fácil começar e navegar por quaisquer desafios que surgirem durante o desenvolvimento.

5.2 Arquitetura do LangChain

A arquitetura do LangChain pode ser entendida em três componentes principais:

Camada de Abstração: Esta camada fornece uma interface simples e unificada para interagir com diferentes LLMs. Ele abstrai as complexidades de carregar, inicializar e executar previsões em modelos, oferecendo uma API consistente independentemente do modelo subjacente.

Camada de Processamento de Texto: O LangChain inclui ferramentas robustas para processamento de texto, incluindo tokenização, análise sintática, reconhecimento de entidades nomeadas (NER) e muito mais. Esta camada prepara os dados de entrada e saída para que possam ser processados de forma eficaz pelos modelos de linguagem.

Camada de Modelo: Aqui é onde os próprios LLMs residem. O LangChain suporta uma variedade de modelos de linguagem, desde modelos pré-treinados de uso geral até modelos personalizados específicos de domínio. Esta camada lida com a execução de previsões, gerenciamento de recursos computacionais e interação com as APIs dos modelos.

5.3 Exemplo Básico: Consultando um LLM

Vamos ver um exemplo simples de como usar o LangChain para consultar um LLM e obter uma resposta. Neste exemplo, usaremos um modelo de linguagem de uso geral, como o GPT-3, para responder a uma pergunta.

Primeiro, importe as bibliotecas necessárias e configure o cliente LangChain. Em seguida, carregue o modelo de linguagem desejado. Para este exemplo, usaremos o modelo GPT-3. Agora, você pode usar o modelo para fazer uma consulta. Vamos perguntar ao modelo sobre a capital da França.

Instale as dendências necessárias:

```
$ pip install langchain transformers torch langchain_community
```

Código-Fonte:

```
from langchain.prompts import PromptTemplate
from langchain.llms import HuggingFacePipeline
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline

# 1. Baixa e configura o modelo localmente
model_name = "distilgpt2"  # Modelo pequeno e leve
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# 2. Cria um pipeline de geração de texto
pipe = pipeline(
 "text-generation",
 model=model,
 tokenizer=tokenizer,
 max_new_tokens=50  # Limita o tamanho da resposta
)
```

```
# 3. Integra o pipeline com o LangChain
llm = HuggingFacePipeline(pipeline=pipe)

# 4. Define um template de prompt
prompt = PromptTemplate(
 input_variables=["pergunta"],
 template="Responda de forma clara e concisa: {pergunta}"
)

# 5. Cria a cadeia (chain)
chain = prompt | llm

# 6. Executa a cadeia com uma pergunta
resposta = chain.invoke({"pergunta": "Qual é a capital da França?"})
print(resposta)

print(response.text)
```

Explicação:

- Dependências: Você precisa instalar langchain, transformers e torch (PyTorch, necessário para rodar o modelo).
- Modelo: O distilgpt2 é uma versão destilada e leve do GPT-2, ideal para testes locais. Ele será baixado automaticamente na primeira execução (cerca de 300 MB).
- Pipeline: O transformers.pipeline configura o modelo para geração de texto.
- HuggingFacePipeline: Integra o pipeline com o LangChain.
- Prompt e Chain: Como no exemplo anterior, criamos um prompt e uma cadeia.
- Execução: A pergunta é processada localmente pelo modelo.

Saída esperada:

- A resposta pode variar um pouco devido à natureza probabilística do modelo, mas seria algo como:
- "Responda de forma clara e concisa: A capital da França é Paris."

Requisitos:

- Hardware: Funciona em CPUs comuns, mas uma GPU acelera o processo (não é obrigatória).
- Espaço: O modelo ocupa cerca de 300 MB no disco. Internet: Necessária apenas na primeira execução para baixar o modelo.

Personalização:

- Se quiser um modelo diferente, substitua distilgpt2 por outro nome de modelo da Hugging Face (ex.: gpt2, facebook/opt-125m), mas verifique os requisitos de memória.
- Ajuste max_new_tokens para respostas mais curtas ou longa

Este exemplo básico demonstra a simplicidade de usar o LangChain para interagir com LLMs. No entanto, o LangChain oferece muito mais recursos e funcionalidades para construir aplicativos de chatbot mais robustos.

Referências Bibliográficas

[1] Sameera A. Abdul-Kader and John Woods. Survey on Chatbot Design Techniques in Speech Conversation Systems. *International Journal of Advanced Computer Science and Applications*, 6(7):72–80, 2015.

[2] Everton Moschen Bada. Uma proposta para extracao de perguntas e respostas de textos. *XVII Congreso Internacional de Informatica Educativa, TISE*, pages 44–49, 2012.

[3] Bhriguraj Borah, Dhrubajyoti Pathak, and Priyankoo Sarmah. Survey of Text based Chatbot in Perspective of Recent Technologies. In *International Conference on Computational Intelligence, Communications, and Business Analytics. CICBA 2018: Computational Intelligence, Communications, and Business Analytics*, volume 1031, pages 84–96. Springer Singapore, 2019.

[4] Noam Chomsky and David W Lightfoot. *Syntactic structures*. Walter de Gruyter, 2002.

[5] Arandir Cordeiro da Silva and Evandro De Barros Costa. Um Chatterbot aplicado ao Turismo: Um estudo de caso no Litoral Alagoano e na Cidade de Maceió. pages 160–167, 2007.

[6] Ryan Ribeiro de Azevedo. *Um sistema de dialogo inteligente baseado em logica de descricoes*. PhD thesis, 2015.

[7] Giovanni De Gasperis, Isabella Chiari, and Niva Florio. *AIML knowledge base construction from text corpora*, volume 427. 2013.

[8] David Ferrucci. This Is Watson. *Journal of Research and Development*, 56(3):88, 2012.

[9] Sviatlana Höhn. *Artificial Companion for Second Language Conversation*. 2019.

[10] Jacobstein, Murray, Sams, and Sincoff. A multi-agent associate system guide for a virtual collaboration center. *Proceedings of the International Conference on Virtual Worlds and Simulation Conference*, pages 215–220, 1998.

[11] Antonio Fernando Lavareda Jacob Junior. *Buti: um Companheiro Virtual baseado em Computacao Afetiva para Auxiliar na Manutencao da Saude Cardiovascular*. PhD thesis, 2008.

[12] Lorenz Cuno Klopfenstein, Saverio Delpriori, Silvia Malatini, and Alessandro Bogliolo. The Rise of Bots: A Survey of Conversational Interfaces, Patterns, and Paradigms. *Proceedings of the 2017 Conference on Designing Interactive Systems*, pages 555–565, 2017.

[13] Aliane Loureiro Krassmann, Fabrício Herpich, Álvaro Souza Pereira da Silva, Anita Raquel da Silva, Cristiane de Souza Abreu, Marcelo Augusto Rauh Schmitt, Magda Bercht, and Liane Margarida Rockenbach Tarouco. FastAIML: uma ferramenta para apoiar a geracao de base de conhecimento para chatbots educacionais. *Revista Novas Tecnologias na Educacao (RENOTE)*, 15(2):1–10, 2017.

[14] Helton Kraus and Anita Fernandes. Ivetebyte: Um Chatterbot para Area Imobiliaria Integrando Raciocinio Baseado em Casos. *Revista Iberica de Sistemas e Tecnologias de Informacao*, (1):40–51, 2008.

[15] Rupert Lane, Anthony Hay, Arthur Schwarz, David M. Berry, and Jeff Shrager. ELIZA Reanimated: The world's first chatbot restored on the world's first time sharing system. pages 1–21, 2025.

[16] Michelle Denise Leonhardt, Ricardo Neisse, and Liane Margarida Rockenbach Tarouco. MEARA: Um Chatterbot Temático para Uso em Ambiente Educacional, 2003.

[17] Jiwei Li, Will Monroe, Tianlin Shi, Sèbastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial Learning for Neural Dialogue Generation. pages 2157–2169, 2018.

[18] Carlos Eduardo Teixeira Lima Lima. *Um Chatterbot para Criacao e Desenvolvimento de Ontologias com Logica de Descricao*. PhD thesis, 2017.

[19] Rafael Luiz De Macedo and Elvis Fusco. An Intelligent Robotic Engine Using Digital Repository of the DSpace Platform. (c):17–23, 2014.

[20] S. Madhumitha, B. Keerthana, and B. Hemalatha. Interactive Chatbot Using AIML. *International Journal Of Advanced Networking And Applications*, pages 217–223, 2015.

[21] Francisco Supino Marcondes, Jose Joao Almeida, and Paulo Novais. A Short Survey on Chatbot Technology : Failure in Raising the State of the Art. pages 28–36, 2020.

[22] Luís Antonio Marcuschi. *Análise da Conversação*. 1986.

[23] Filomena Maria, Gonçalves Da, Silva Cordeiro, and Rodrigo Lins Rodrigues. Um ambiente virtual de aprendizagem apoiado por um agente virtual: chatterbot. *Encontro Internacional TIC e Educacao 2010 Lisboa Portugal*, 2010.

[24] Michael L Mauldin. CHATTERBOTS, TINYMUDS, and the Turing Test Entering the Loebner Prize Competition. *AAAI*, 94:16–21, 1994.

[25] André M. M. Neves and Flávia de Almeida Barros. iAIML: Um mecanismo para tratamento de intencao em chatterbots. *Congresso da Sociedade Brasileira de Computacao*, pages 1032–1041, 2005.

[26] Alex Fernando Teixeira PRIMO and Luciano Roth COELHO. A chatterbot Cybelle: experiência pioneira no Brasil. *Mídia, textos e contextos. Porto Alegre: EDIPUCRS*, pages 259–276, 2001.

[27] Sumit Raj. *Building Chatbots with Python*. 2019.

[28] Kiran Ramesh, Surya Ravishankaran, Abhishek Joshi, and K. Chandrasekaran. A Survey of Design Techniques for Conversational Agents. In *International Conference on Information, Communication and Computing Technology*, volume 835, pages 336–350, 2017.

[29] Stuart Russel and Peter Norving. *Inteligência Artificial*. 2013.

[30] Ayesha Shaikh, Geetanjali Phalke, Pranita Pat, Sangita Bhosale, and Jyoti Raghatwan. A Survey On Chatbot Conversational Systems. *International Journal of Engineering Science and Computing*, 6(11):3117–3119, 2016.

[31] Moolchand Sharma, Shivang Verma, and Lakshay Sahni. Comparative Analysis of Chatbots. *SSRN Electronic Journal*, 2020.

[32] Heung-yeung Shum, Xiao-dong He, and Di Li. From Eliza to XiaoIce: challenges and opportunities with social chatbots. *Frontiers of Information Technology e Electronic Engineering*, 19(1):10–26, 2018.

[33] Paula Geralda Barbosa Coelho Torreao. *Project Management Knowledge Learning Environment: Ambiente Inteligente de Aprendizado para Educacao em Gerenciamento de Projetos*. PhD thesis, 2005.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):5999–6009, 2017.

[35] R Wallace. The Elements of AIML Style. *Alice AI Foundation, Inc*, pages 12–77, 2003.

[36] Richard S Wallace. Don' t Read Me: A.L.I.C.E. and AIML Documentation. pages 1–72, 2000.

[37] Richard S. Wallace. The anatomy of A.L.I.C.E. *Parsing the Turing Test. Springer, Dordrecht*, pages 181–210, 2009.

[38] Joseph Weizenbaum. ELIZA - A Computer Program For the Study of Natural Language Communication Between Man And Machine. *Communications of the ACM*, 9(1), 1966.

[39] Hiroshi Yamaguchi, Maxim Mozgovoy, and Anna Danielewicz-Betz. A Chatbot Based On AIML Rules Extracted From Twitter Dialogues. *Communication Papers of the 2018 Federated Conference on Computer Science and Information Systems*, 17:37–42, 2018.