Capítulo

6

Chabots Inteligentes na Saúde: Implementações com Modelos Abertos e Dados Próprios

Fabio Martiniano Sato, Júlio Eduardo Silva, Rogério de Oliveira

6.1. Introdução

Chatbots inteligentes são sistemas de software avançados projetados para simular interações humanas por meio de texto ou fala, utilizando técnicas sofisticadas de inteligência artificial e processamento de linguagem natural. Esses sistemas têm demonstrado eficácia ao responder perguntas, oferecer informações e realizar tarefas automatizadas em diversos contextos. A base para essa eficácia reside no uso de grandes modelos de linguagem, ou LLMs (Large Language Models), como o GPT-4 e o BERT, modelos de redes neurais profundas treinados com grandes volumes de dados para compreender e gerar linguagem natural de forma precisa.

A criação e o treinamento desses modelos podem ser exigentes em termos de hardware e dados, resultando em custos elevados e longos tempos de processamento. Contudo, modelos abertos oferecem uma alternativa viável ao permitir adaptação para necessidades específicas por meio de ajuste fino. Esse processo permite personalizar modelos pré-treinados, melhorando seu desempenho em contextos específicos e reduzindo os custos em comparação com a criação de novos modelos do zero.

Na área da saúde, a aplicação de LLMs está promovendo uma transformação significativa, com impacto direto em práticas clínicas e médicas. Esses modelos são capazes de processar vastas quantidades de dados e lidar com a complexidade associada, elevando a qualidade dos serviços médicos, reduzindo a carga de trabalho dos profissionais e ampliando o acesso aos cuidados. As aplicações mais comuns incluem suporte à tomada de decisões clínicas, automação de relatórios e laudos, triagem de pacientes e educação em saúde (Zhou et al. 2023). No entanto, o uso desses modelos enfrenta desafios significativos em relação a questões éticas, privacidade dos pacientes e a necessidade de garantir que as recomendações sejam seguras e confiáveis, elementos fundamentais para a aceitação e efetividade dessas tecnologias no setor.

Este capítulo explora o desenvolvimento de chatbots inteligentes, com base no uso de LLMs, voltados para a área da saúde. A ênfase é no uso de modelos abertos e dados próprios. Ao longo do capítulo destacam-se os principais conceitos envolvidos nos grandes modelos de linguagem e na sua personalização para temas específicos, como o da saúde, e são apresentadas as principais abordagens que podem empregadas. A aplicação dessas técnicas é exemplificada com o uso do modelo aberto LLaMA-3-8B (AI@Meta 2024) na construção de chatbots em três cenários distintos: empregando o mo-delo sem qualquer ajuste; personalizando o modelo com um único documento (com diretrizes para profilaxia de infecção pelo HIV); e, finalmente, utilizando um grande conjunto de dados extraído da Wikipedia de temas voltados à saúde (Coronavírus, HIV, Câncer e Diabetes).

O restante deste capítulo está organizado da seguinte forma: a seção 6.2 explora a evolução e os fundamentos dos grandes modelos de linguagem, abordando o conceito de ajuste fino, frameworks de desenvolvimento e suas aplicações no contexto da saúde. Em seguida, a seção 6.3 apresenta e discute a implementação de chatbots inteligentes em três abordagens distintas de uso dos LLMs, destacando suas vantagens e desafios. Por fim, a seção 6.4 examina os resultados obtidos, suas limitações e o potencial futuro dessas aplicações para o setor.

Assim, este capítulo busca oferecer uma visão abrangente das oportunidades e desafios associados ao uso de LLMs na saúde, ressaltando o papel do ajuste fino na personalização desses modelos e as questões éticas envolvidas. Dessa forma, espera-se que o leitor tenha uma compreensão aprofundada tanto das tecnologias subjacentes quanto das implicações práticas e éticas do uso de LLMs neste campo.

6.2. LLMs, os grandes modelos de linguagem

Há muito tempo empregam-se sistemas que tratam a linguagem natural, mas só nos últimos anos esses sistemas parecem ter atingido uma capacidade de um potencial disruptivo sem precedentes. Os recentes grandes modelos de linguagem (LLMs) são hoje a base de sistemas que requerem compreender e processar a linguagem natural, com aplicações na transcrição da fala e tradução de textos, automação de tarefas por assistentes virtuais e, mais recentemente, a geração de textos com fluidez humana. Esses avanços têm permitido a criação de chatbots cada vez mais inteligentes, que podem simular conversas humanas e fornecer suporte valioso em diversas áreas.

6.2.1. Breve histórico

O campo do processamento de linguagem natural (PLN) apresenta uma evolução notável ao longo dos anos. Desde os modelos estatísticos predominantes no início dos anos 2000, o PLN avançou significativamente para o desenvolvimento dos sofisticados grandes modelos de linguagem (LLMs) que se conhecem atualmente (ver Figura 6.1). Esta evolução ampliou as capacidades dos sistemas de PLN e consolidou o campo como uma área de pesquisa essencial e altamente dinâmica, impulsionando inovações com impacto direto em diversas aplicações, incluindo chatbots inteligentes na área da saúde.

Em 2003, Ronald Rosenfeld fez contribuições significativas para o avanço dos modelos estatísticos, ao introduzir técnicas como *N*-grams e interpolação, que

aprimoraram a precisão dos modelos empregados em sistemas de reconhecimento de fala e abriram caminho para aplicações mais sofisticadas. Simultaneamente, pioneiros do aprendizado profundo, como Geoffrey Hinton e Yoshua Bengio, estabeleceram os fundamentos dos modernos sistemas de linguagem. No início da década de 2010, surgiram os primeiros modelos de linguagem baseados em redes neurais recorrentes (RNNs) e redes neurais convolucionais (CNNs), marcando o início de um novo patamar e paradigma no campo dos modelos de linguagem. Desde então, os avanços têm ocorrido a um ritmo cada vez mais acelerado.

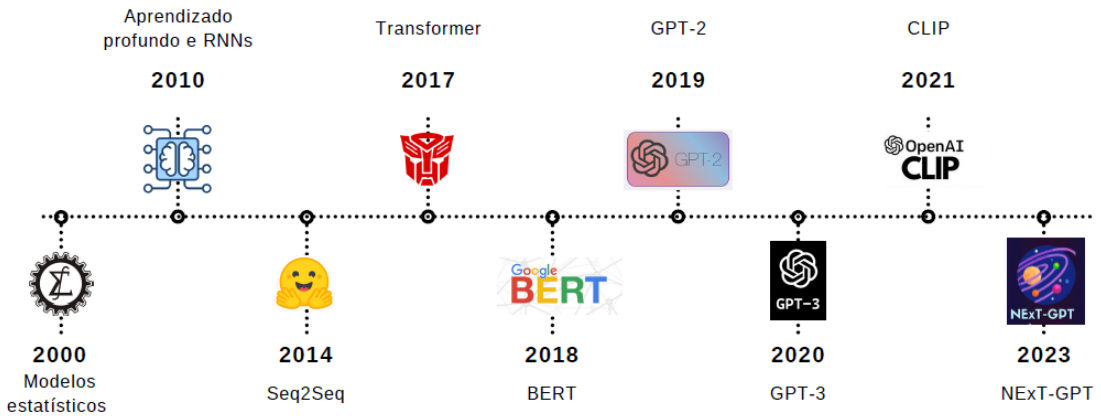

Figura 6.1: Linha do tempo que ilustra os marcos históricos sobre modelos de linguagem.

Em 2014, Sutskever et al. 2014 publicou o artigo "Sequence to Sequence Learning with Neural Networks", no qual introduziu o modelo Seq2Seq. Este modelo foi pioneiro na aplicação de redes neurais recorrentes para a tradução de sequências de palavras entre diferentes idiomas. Em 2017, Vaswani et al. 2017 trouxe uma inovação disruptiva com o artigo "Attention is All You Need", que apresentou a arquitetura Transformer. Esta nova arquitetura eliminou a necessidade de redes neurais recorrentes e permitiu o processamento paralelo de cadeias de palavras, estabelecendo-se como a base predomi-nante para os modelos de linguagem atuais. Em 2018, Devlin et al. 2018 introduziu o BERT (Bidirectional Encoder Representations from Transformers), inovando ao adotar uma abordagem de pré-treinamento bidirecional com Transformers. Em 2019, a OpenAI lançou o GPT-2 (Generative Pre-trained Transformer 2), também baseado na arquitetura Transformer. O GPT-2 destacou-se por sua capacidade de popularizar o uso de modelos de linguagem em larga escala para responder a perguntas em linguagem natural, e foi seguido pelas versões GPT-3 e GPT-4, que apresentaram ainda mais avanços em capacidade e parâmetros. Em 2021, a OpenAI apresentou o CLIP (Contrastive Language-Image Pre-training), um modelo multimodal que compreende tanto texto quanto imagens. Mais recentemente, (Wu et al. 2023) propôs o modelo NExT-GPT, que além de permitir entradas multimodais, também gera saídas em diversos formatos, incluindo texto, áudio, imagem e vídeo.

6.2.2. Princípios dos grandes modelos de linguagem

Modelos de linguagem são sistemas probabilísticos projetados para modelar a distribuição conjunta de tokens — unidades de texto que podem variar desde letras e palavras até fragmentos dessas unidades — em grandes corpora de

documentos, como páginas da web e publicações, criando representações detalhadas e úteis desses dados textuais.

Atualmente, os grandes modelos de linguagem são predominantemente baseados em redes neurais profundas. O treinamento desses modelos envolve ajustar os pesos de redes neurais com um número extremamente grande de camadas e parâmetros para representar com precisão os dados textuais. Essa abordagem possibilita que os modelos aprendam e se adaptem de maneira eficiente a uma ampla gama de tarefas, refletindo a evolução contínua e a sofisticação crescente dos métodos de processamento de linguagem natural.

As redes neurais recorrentes, particularmente as LSTMs (Long Short-Term Memory, ver Hochreiter and Schmidhuber 1997), desempenhavam um papel crucial na modelagem de linguagem até meados de 2017 (Vaswani et al. 2017). Essas redes eram projetadas para capturar como as palavras anteriores influenciam o significado das palavras subsequentes, exemplificado pela ambiguidade da palavra "manga" em diferentes contextos — como em "A camisa tinha uma manga longa" versus "Comi a manga no café da manhã". No entanto, as LSTMs apresentavam limitações significativas, como alto custo computacional e restrições ao processamento paralelo.

Essas limitações evidenciaram a necessidade de novos métodos mais eficientes, resultando no desenvolvimento da arquitetura *transformers* (Vaswani et al. 2017). A arquitetura *transformers* aborda essas deficiências ao eliminar o processamento sequencial e permitir o processamento paralelo, o que melhora substancialmente a escalabilidade e a eficiência no treinamento dos modelos.

Atualmente, para a construção de aplicações que utilizam grandes modelos de linguagem, entender a arquitetura *transformers* é crucial. Embora um conhecimento profundo dos detalhes técnicos não seja obrigatório, entender os princípios fundamentais dessa arquitetura proporciona uma visão mais clara das capacidades, diferenças e usos dos diversos modelos de linguagem modernos.

6.2.3. Tokens, Embeddings, Transformers e o Mecanismo de Atenção

Os LLMs, como modelos de redes neurais profundas, são basicamente modelos que processam entradas por sucessivas camadas de neurônios (os elementos) da rede. No caso da arquitetura transformers há basicamente dois conjuntos de camadas distintas, a de *encode* e a de *decode*. A primeira cria representações internas de textos de entrada, e a segunda produz saídas de texto a partir de representações internas. A Figura 6.2, mostra uma representação simplificada dessa arquitetura omitindo uma série de detalhes como camadas de normalização, múltiplas *cabeças de atenção* empregadas etc. mas suficientemente útil para os propósitos desta introdução.

O *mecanismo de atenção* é um elemento crucial dentro da arquitetura *transformers*, permitindo a relação entre diferentes elementos do texto — como termos e palavras representados por *tokens* — para criar uma representação única sem depender de recorrências ou convoluções (Vaswani et al. 2017). Esta representação vetorial dos textos é fundamental para os modelos de linguagem, e sua construção será detalhada e exemplificada nas seções subsequentes.

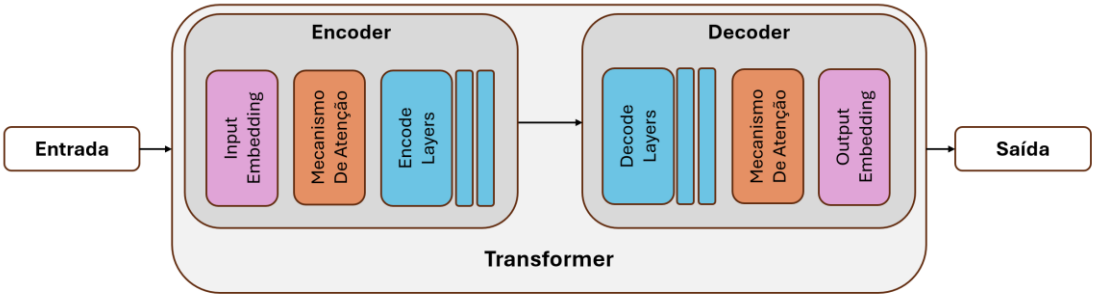

Figura 6.2: Representação simplificada das camadas da arquitetura Transformer, enfatizando as camadas de Encode/Decode e o Mecanismo de Atenção.

A. Tokens. No processamento de linguagem natural, é fundamental que o texto ou as palavras sejam representados de forma vetorial, através de *embeddings*. Durante o processamento do modelo, são geradas diversas representações vetoriais para cada entrada. A tokenização é uma etapa crítica na construção desses embeddings, pois divide o texto em unidades menores conhecidas como tokens. Dependendo do modelo, esses tokens podem ser palavras, subpalavras, caracteres ou símbolos, e cada modelo pode usar um dicionário de tokens distinto (para mais detalhes, ver Mielke et al. 2021). Devido à diversidade dos tokens em diferentes idiomas, é necessário utilizar um modelo específico para a língua em questão ou um modelo multilingue para lidar com múltiplas línguas simultaneamente.

```
neuralmind/bert-large-portuguese-cased
TOKENS...

Frase:       O gato subiu no telhado.
Tokens:      [101, 231, 15997, 10996, 202, 16267, 119, 102]
Tokens decodificados:    ['[CLS]', 'O', 'gato', 'subiu', 'no', 'telhado', '.', '
    [SEP]']

FacebookAI/xlm-roberta-base
TOKENS...

Frase: O gato subiu no telhado.
Tokens: [0, 180, 125583, 1614, 3214, 110, 120, 178261, 5, 2]
Tokens decodificados:    ['<s>', ' O ', ' gato ', ' sub ', 'iu', ' no ', '
      ta ', 'lhado', '.', '</s>']
```

Código Fonte 6.1: (saída) Tokens para a frase *O gato subiu no telhado.* com BERTimbau e RoBERTa. BERTimbau usa tokenização direta com palavras inteiras e tokens especiais como CLS e SEP enquanto RoBERTa usa tokenização por subpalavras, com tokens especiais como <s> e </s>.

Os modelos apresentados diferem significativamente em seus dicionários de tokens: o BERTimbau utiliza um dicionário com 29.794 tokens, enquanto o RoBERTa possui um dicionário de 250.002 tokens. Essa diferença reflete a diversidade e a granularidade dos tokens que cada modelo emprega. O tamanho do dicionário pode impactar diretamente o desempenho do modelo, uma vez que modelos com um maior número de tokens têm a capacidade de representar uma gama mais ampla de variantes linguísticas e contextos. No entanto, o tamanho do dicionário é apenas um dos muitos fatores que influenciam o desempenho do modelo; outros aspectos como o conjunto de dados de treinamento, o tamanho dos embeddings e a complexidade da arquitetura da rede neural também desempenham papéis cruciais.

Além dos tokens comuns, existem tokens especiais que têm funções específicas, como indicar o início ou o fim de uma sequência. Por exemplo, o token [MASK], presente em alguns modelos como o BERTimbau, é usado para identificar e prever um termo faltante em uma frase, o que é útil em tarefas de preenchimento de lacunas. Essas particularidades na tokenização ajudam a ajustar a capacidade do modelo para tarefas específicas de processamento de linguagem natural.

B. Embedding. Para que um modelo de linguagem compreenda e processe texto de maneira eficaz, cada token deve ser representado por um vetor numérico, conhecido como embedding. Esses vetores são aprendidos durante o treinamento e têm a vantagem de que tokens com significados semelhantes são representados por vetores próximos em um espaço vetorial de alta dimensão. Isso possibilita ao modelo captar relações semânticas e contextuais entre as palavras, facilitando a interpretação e o processamento do texto.

Exemplo. Ainda na frase "*O gato subiu no telhado.*". O Código Fonte 6.2 ilustra como esses embeddings são estruturados para a frase fornecida.

```
neuralmind/bert-large-portuguese-cased
ORIGINAL EMBEDDING

Shape: torch.Size([1, 8, 1024])

Token: [CLS]        Embedding: [-0.0296  -0.0720  -0.0117   0.0729   0.0563]...
Token: O            Embedding: [ 0.3458  0.2267   0.3594  -0.0551    0.5952   ]...
Token: gato         Embedding: [ 0.4060   0.1446   0.0774  -0.2634   0.0110 ]...
Token: subiu        Embedding: [ 0.5057   0.0525  -0.0958   0.0440  -0.771  ]...
Token: no           Embedding: [ 0.1305  -0.1004  -0.1166   0.3667   0.5283   ]...
Token: telhado      Embedding: [ 0.1708  -0.0197   0.4807   0.0514  -0.4222  ]...
Token: .            Embedding: [ 0.0766  -0.4390   0.3397  -0.5754    -0.1227 ]...
Token: [SEP]        Embedding: [-0.3091   0.16961  0.1518   0.0987   -0.0102 ]...
```

Código Fonte 6.2: (saída) Representação de tokens com o modelo BERTimbau. O vetor de embedding para cada token captura características semânticas e sintáticas do token dentro do contexto da frase. Essas representações vetoriais são utilizadas pelo modelo para realizar várias tarefas de processamento de linguagem natural, como compreensão de texto e geração de respostas.

O modelo BERTimbau produz uma representação vetorial (ver exemplo na Figura 6.3) de cada token em um espaço de 1024 dimensões. Essa abordagem simplifica a complexidade da codificação *one-hot* original, que, devido ao dicionário de 29.794 tokens, seria significativamente mais extensa. Em contraste, o modelo RoBERTa utiliza vetores de embedding com 768 dimensões, proporcionando uma representação ainda mais compacta, apesar de seu dicionário ser muito maior, com 250.002 tokens. Embora os embeddings variem entre diferentes modelos, no mesmo modelo, o token “gato” mantém uma representação consistente, independentemente da frase em que aparece, como em “O gato subiu no telhado” ou “O gato é preto”.

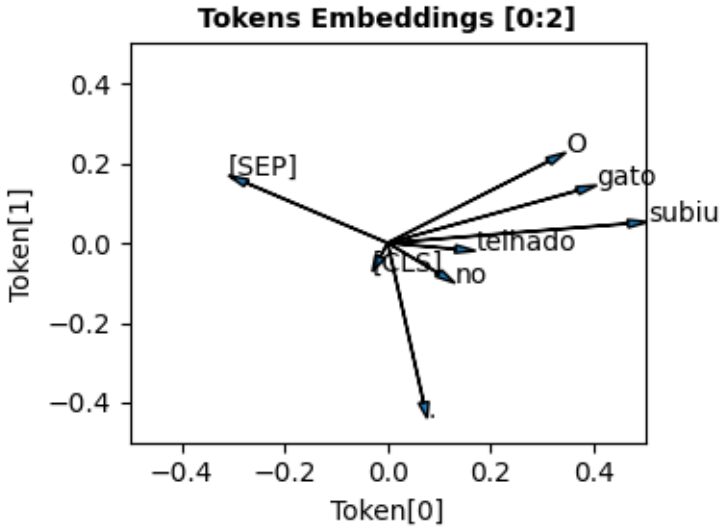

Figura 6.3: Representação vetorial dos tokens em um espaço de alta dimensão. Para facilitar a visualização, apenas as duas primeiras dimensões das centenas de dimensões utilizadas pelos modelos.

Depois de definir os embeddings iniciais dos tokens, o próximo passo na arqui-tetura dos *transformers* é calcular novos embeddings que incorporam a interação entre todos os tokens do texto. Esse processo considera a dependência de um token em relação aos outros. Em termos matemáticos, o novo embedding $E_{gato}[i+1]$ é dado por:

$$E_{gato}[i+1] = \sum_{t=\text{all tokens}} A_t[i] E_t[i]$$

onde $E_t[i]$ representa sucessivos embeddings e $A_t[i]$ é o vetor de atenção de cada token *t*.

Diferentemente do embedding *inicial*, que é consistente para a palavra "gato" tanto na frase "O gato subiu no telhado" quanto em "O gato é preto", o novo embedding varia conforme o contexto em que o termo aparece. Isso ocorre porque ele é ajustado com base nas interações e relações contextuais entre todos os tokens na sequência. Essa capacidade de adaptar o embedding de um token de acordo com seu contexto é um dos principais avanços e vantagens da arquitetura *transformers*.

C. Atenção. O mecanismo de atenção é importante na arquitetura dos *transformers*, pois permite que o modelo se concentre em diferentes partes da entrada ao processar cada token, o que facilita a modelagem de dependências de longo alcance entre os tokens em uma sequência. Por exemplo, na frase "O gato subiu no telhado", espera-se que o token "O" esteja mais fortemente associado ao token "gato" do que a qualquer outro token, e que "telhado" esteja mais relacionado ao verbo "subiu" do que aos demais termos. Esses pesos de atenção, que indicam a força das associações entre as palavras, são aprendidos

durante o treinamento do modelo. Com base nesses pesos, o modelo calcula uma nova representação (embedding) para cada token.

Exemplo. O Código Fonte 6.3 mostra os pesos de atenção gerados para cada token na frase “O gato subiu no telhado”, utilizando o modelo BERTimbau. As Figuras 6.4 e 6.5 fornecem uma representação gráfica desses resultados, ilustrando as associações aprendidas pelo modelo de maneira visual.

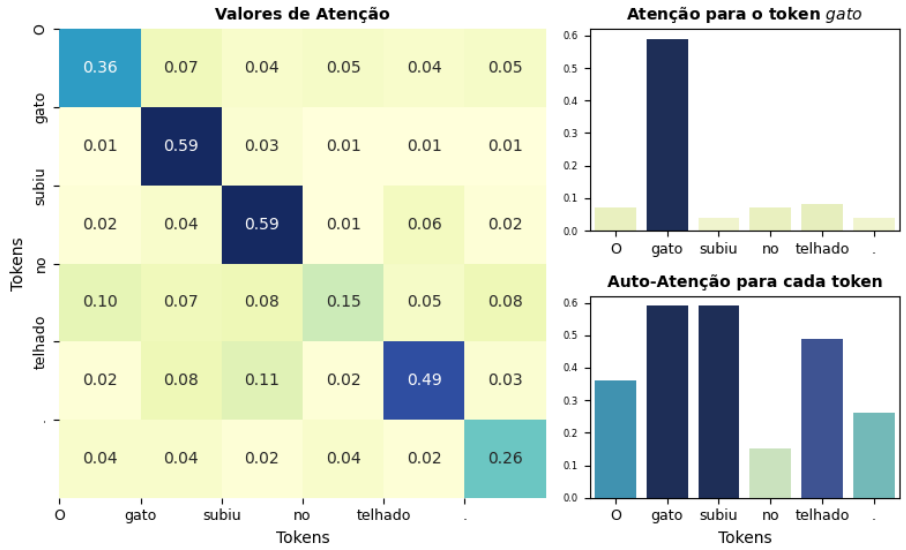

Figura 6.4: Visualização dos pesos de atenção gerados para cada token na frase "O gato subiu no telhado” utilizando o modelo BERTimbau. À esquerda, a matriz completa de atenção mostra como cada token presta atenção a todos os outros tokens. À direita, a parte superior exibe os valores de atenção especificamente para o token "gato”, enquanto a parte inferior mostra os valores de auto-atenção para o mesmo token. Observa-se que gato” tem maior atenção em relação a"subiu” (o verbo principal) e “telhado” (o advérbio), refletindo a importância desses tokens no contexto da frase. Tokens com menores valores de atenção estão listados abaixo.

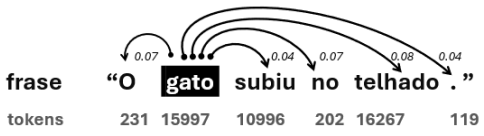

Figura 6.5: Pesos de atenção para o token "gato” na frase"O gato subiu no telhado” uti-lizando o modelo BERTimbau. A figura mostra como o token "gato” se relaciona com os demais tokens na frase, com pesos que indicam a força da dependência entre "gato” e cada um dos outros tokens. Esses pesos refletem o grau de atenção que o token “gato”presta a cada um dos tokens na sequência. Cada vetor de atenção é utilizado para ge-rar novos embeddings dos tokens, incorporando informações contextuais e dependências entre todos os tokens da frase.

```
1 neuralmind/bert-large-portuguese-cased
2 VETORES DE ATENÇÃO
3
4           [CLS]     O  gato  subiu    no  telhado     .  [SEP]
5 [CLS]      0.09  0.07  0.09   0.06  0.08     0.07  0.12   0.42
6 O          0.08  0.36  0.07   0.04  0.05     0.04  0.05   0.31
7 gato       0.19  0.01  0.59   0.03  0.01     0.01  0.01   0.15
8 subiu      0.05  0.02  0.04   0.59  0.01     0.06  0.02   0.21
9 no         0.08  0.10  0.07   0.08  0.15     0.05  0.08   0.39
10 telhado   0.10  0.02  0.08   0.11  0.02     0.49  0.03   0.16
11 .         0.03  0.04  0.04   0.02  0.04     0.02  0.26   0.54
12 [SEP]     0.02  0.00  0.01   0.00  0.00     0.00  0.01   0.96
13
14 # valores arredondados para 2 casas decimais
```

Código Fonte 6.3: (saída) Matriz de pesos de atenção para a frase "O gato subiu no telhado" gerada pelo modelo BERTimbau. Cada célula na matriz mostra o grau de atenção que um token (linha) presta a outro token (coluna) na sequência. Por exemplo, o valor na célula correspondente à linha "gato" e à coluna "subiu" indica a importância da palavra "subiu" para a representação de "gato". Valores arredondados para duas casas decimais.

D. Final Embedding. Após o processamento, cada token possui uma nova representação vetorial que reflete as informações contextuais e as relações entre todos os tokens na sequência. Esta nova representação é dinâmica, ajustando-se conforme o contexto específico em que o token aparece, o que permite uma compreensão mais rica e contextualizada.

Exemplo. O Código Fonte 6.4 ilustra a representação final dos embeddings dos tokens para a frase "O gato subiu no telhado" no modelo BERTimbau. Esse processo pode envolver múltiplas camadas de processamento; por exemplo, o BERTimbau utiliza "num_hidden_layers": 24 e "num_attention_heads": 16, que são responsáveis pelo processamento paralelo das informações. O essencial é que, ao final, é gerada uma representação que considera e pondera as interações entre os diferentes termos no texto.

```
1 neuralmind/bert-large-portuguese-cased
2 FINAL EMBEDDING
3
4 Shape: torch.Size([1, 8, 1024])
5
6 Token: [CLS]        Embedding: [ 0.1447 -0.3951  -0.2720 -0.2258 -0.155 ]...
7 Token: O            Embedding: [ 0.5651  0.5572  -0.1385  0.1579 -0.5751]...
8 Token: gato         Embedding: [ 0.9112  0.5293   0.0105 -0.4229  0.8343 ]...
9 Token: subiu        Embedding: [ 0.6923  0.8415 -0.2350   0.0851 -0.0757]...
10 Token: no          Embedding: [ 0.5188  0.4285 -0.1618   0.4793 -0.7100]...
11 Token: telhado     Embedding: [ 0.4686  0.8759 -0.5351  -0.3785  0.7844]...
12 Token: .           Embedding: [ 0.3682  0.7553  -0.3403  0.4075  0.1020]...
13 Token: [SEP]       Embedding: [ 0.9630  0.5064 -0.6537   0.0168 -0.1162]...
14
15 # apenas os 5 primeiros valores de cada vetor é exibido
```

Código Fonte 6.4: (saída) Representação final dos tokens para a frase "O gato subiu no telhado" gerada pelo modelo BERTimbau. Cada token é representado por um vetor de embedding de 1.024 dimensões, que incorpora informações contextuais e relacionais com outros tokens na frase. Os vetores exibidos são apenas uma amostra, mostrando os primeiros cinco valores de cada vetor de embedding, para ilustrar a complexidade e a riqueza das representações finais produzidas pelo modelo.

E. Predict, mask. Considere a frase incompleta: "O gato subiu no [MASK].", onde [MASK] representa a palavra (ou token) que desejamos prever. Utilizando o modelo e a representação vetorial discutidos anteriormente, é possível identificar os tokens com maior probabilidade de preencher o espaço da máscara.

Exemplo. O Código Fonte 6.5 ilustra a aplicação do modelo BERTimbau para prever a *próxima palavra* na frase "O gato subiu no ?". Neste processo, o modelo usa o token especial [MASK] para indicar a posição a ser preenchida. É importante notar que nem todos os modelos de linguagem utilizam o token [MASK]. Além disso, modelos diferentes podem empregar estratégias variadas para essa tarefa. Por exemplo, o modelo RoBERTa utiliza um método diferente para preencher lacunas, o que pode influenciar os resultados da previsão.

```
1 nmauralmind/bert-large-portuguese-cased
2 Predict next word...
3
4    score  token token_str                  sequence
5 0   0.36  16267   telhado  O gato subiu no telhado.
6 1   0.08   8105      chão     O gato subiu no chão.
7 2   0.07   3883     carro    O gato subiu no carro.
8 3   0.03  16909      muro     O gato subiu no muro.
9 4   0.03   5340     palco    O gato subiu no palco.
10
11   score  token token_str                  sequence
12 0   0.36  16267   telhado  O gato subiu no telhado.
```

Código Fonte 6.5: (saída) Resultados da predição de palavras do modelo BERTimbau para preencher a lacuna na frase "O gato subiu no ?". O modelo utiliza o token especial [MASK] para prever as palavras mais prováveis que podem ocupar o lugar da máscara. A tabela apresenta a pontuação de probabilidade associada a cada token candidato, com a previsão mais provável ("telhado") recebendo a maior pontuação (0.36), seguida por outras possíveis palavras como "chão", "carro", "muro" e "palco".

Apesar das simplificações apresentadas, que não abordam aspectos mais detalhados como as camadas de processamento, paralelismo, funções de ativação, tamanho das sequências e técnicas de normalização, a introdução desses conceitos fornece uma base sólida para compreender o funcionamento dos grandes modelos de linguagem. Para uma exploração mais aprofundada, incluindo exemplos práticos e resultados dos modelos BERTimbau, RoBERTa e TwHIN-BERT, consulte o repositório mencionado no [GitHub LLM], que contém os códigos e saídas apresentados aqui.

6.2.4. Arquiteturas transformers e tarefas

Os transformers surgiram como uma arquitetura *encode-decode* (Vaswani et al. 2017), com aplicações principais na tradução de textos entre diferentes idiomas (por exemplo, codificar um texto em português para uma representação interna e decodificá-lo para o inglês). Desde então, a aplicação dos transformers se expandiu, e eles são atualmente utilizados em diversos tipos de modelos de linguagem. Esses modelos podem ser classificados em três categorias principais com base na estrutura das camadas: *encode*, *decode* e *encode-decode*. Cada uma dessas arquiteturas possui características específicas que as tornam mais adequadas para diferentes tarefas de processamento de linguagem natural, tais como classificação

de textos, Perguntas e Respostas, Geração Automática de Texto, Tradução, Sumarização e Diálogo Interativo.

6.2.5. Modelos pré-treinados e ajuste fino

Os grandes modelos de linguagem (LLMs) podem ser adaptados e personalizados para executar tarefas específicas em diferentes contextos. O ajuste fino é uma técnica comum para essa personalização, que adapta um modelo previamente treinado para atender a novas necessidades em domínios específicos. Para uma revisão abrangente sobre a aplicação do ajuste fino na medicina e seu potencial de personalização, é recomendável consultar Zhou et al. 2023.

A Figura 6.6 ilustra as duas principais estratégias para a adaptação de grandes modelos de linguagem: o treinamento do "zero" e o ajuste fino de modelos gerais. Essas abordagens possibilitam a criação de modelos especializados para diversos domínios, como a saúde. A seguir, detalha-se cada uma dessas estratégias, começando pela construção de modelos desde o "zero".

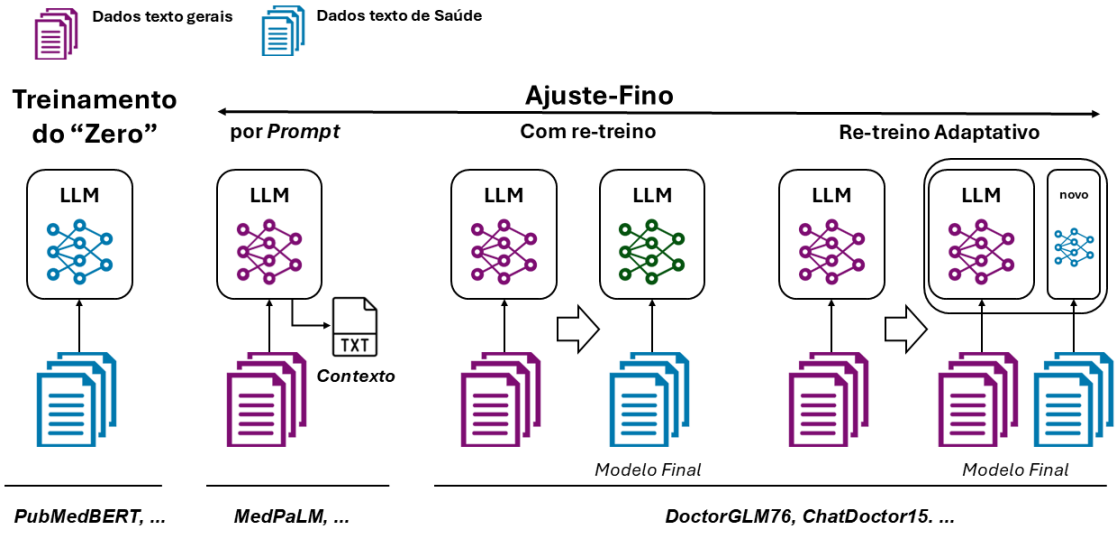

Figura 6.6: Grandes modelos de linguagem para a saúde podem ser desenvolvidos a partir do treinamento do "zero", ou utilizando modelos de linguagem gerais, aos quais são aplicadas técnicas de ajuste fino para adaptá-los ao domínio específico da saúde.

A. Treinamento do "zero". Um LLM para saúde pode ser construído do "zero", treinando o modelo diretamente com um grande conjunto de documentos e dados da área. Nesse caso, o modelo, ou mais especificamente sua rede neural, inicia sem aprendizado prévio, e os pesos são inicializados aleatoriamente para o treinamento com os novos dados. Um exemplo dessa abordagem é o PubMedBERT, criado e treinado com a coleção PubMed de publicações médicas. Embora essa abordagem resulte em modelos altamente eficientes e especializados, ela é dispendiosa, exige grandes bases de dados e enfrenta desafios para atualizar com novos conhecimentos (ver Zhou et al. 2023). Devido ao volume de dados e capacidade de processamento necessários, esses modelos são viáveis

principalmente para grandes empresas ou organizações. Portanto, a construção de modelos dessa forma não será abordada mais detalhadamente aqui.

Uma alternativa mais prática e menos custosa para criar LLMs em um domínio específico é o uso de técnicas de ajuste fino. Nesse método, um grande modelo de linguagem geral, como GPT, LLaMA ou BERT, é ajustado com dados do domínio desejado. Este processo pode incluir ou não o re-treinamento do modelo, o que permite adaptar o modelo a necessidades específicas sem partir do "zero".

B. ajuste fino por Prompt. A técnica de ajuste fino por prompt utiliza instruções específicas ("prompts") para guiar um grande modelo de linguagem geral em tarefas especializadas, sem a necessidade de re-treinamento do modelo. Embora essa abordagem seja menos abrangente do que as técnicas de re-treinamento, oferece uma solução simples e eficiente para situações em que o conjunto de dados disponível é limitado e os recursos computacionais são reduzidos. Na implementação do Modelo II, emprega-se essa técnica ao fornecer uma coleção de documentos a um modelo de linguagem geral e utilizar o aprendizado em contexto (In-Context Learning, ICL) para incorporar novos documentos que não foram incluídos no treinamento original do modelo. Um exemplo dessa abordagem é o MedPaLM, que foi ajustado a partir do modelo geral PalM. Outras técnicas de prompting incluem a Cadeia de Pensamento (Chain-of-Thought, CoT) e a Geração Aumentada de Recuperação (Retrieval-Augmented Generation, RAG). Para uma revisão dessas técnicas e exemplos adicionais na área de saúde, deve-se consultar Zhou et al. 2023.

C. ajuste fino com re-treino. As técnicas de ajuste fino com re-treinamento, frequentemente referidas como ajuste fino ou fine-tuning, envolvem adaptar um modelo de linguagem geral usando um conjunto de dados menor específico para o domínio. Embora esse conjunto seja menor do que o utilizado para o treinamento inicial, ele ainda é maior do que os conjuntos usados nas técnicas de prompt. O re-treinamento ajusta os pesos do modelo original, resultando em uma nova rede neural com pesos ajustados (ver Figura 6.6). Embora essa abordagem seja menos custosa que o treinamento do modelo desde o "zero", ainda exige recursos consideráveis para ajustar bilhões de parâmetros. Exemplos de modelos obtidos por ajuste fino incluem o DoctorGLM76 e o ChatDoctor15, ajustados a partir dos LLMs gerais ChatGLM8,9 e LLaMA4 (Zhou et al. 2023). Devido ao custo de processamento, surgiram alternativas de ajuste fino adaptativo que modificam ou adicionam um pequeno subconjunto de parâmetros ao modelo original, reduzindo a necessidade de re-treinamento completo. Técnicas recentes, como LoRA (Low-Rank Adaptation), Adapter Layers e Prefix-Tuning, têm demonstrado sucesso significativo na adaptação de modelos com recursos limitados. Para uma visão geral dessas técnicas e outras referências, deve-se consultar Naveed et al. 2023. Na implementação do Modelo III, será empregada a técnica de ajuste fino LoRA, que adiciona uma nova camada de transformers ao modelo geral, permitindo o treinamento sem a necessidade de ajustar todos os parâmetros originais (ver figura 6.6).

6.2.6. Modelos em Saúde e Aplicações

As aplicações de grandes modelos de linguagem em medicina e saúde são diversas e abrangem várias áreas, incluindo a sumarização de textos e relatórios clínicos, a

educação em saúde, o diagnóstico e a robótica médica. De acordo com Zhou et al. 2023, essas aplicações podem ser classificadas em duas categorias principais com base no tipo de tarefa que executam:

Tarefas Discriminativas. Nesta categoria, os modelos de linguagem são projetados para realizar tarefas que envolvem a classificação ou a seleção de respostas corretas a partir de um conjunto de opções. Exemplos incluem sistemas que fornecem respostas a perguntas médicas e realizam análises de dados clínicos.

Tarefas Generativas. Os modelos nesta categoria são utilizados para gerar texto ou informações novas com base em entradas fornecidas. Isso inclui a geração de relatórios clínicos, bem como a criação de resumos de prontuários e conteúdo educacional personalizado para profissionais de saúde.

Assim como os LLMs gerais, esses sistemas podem ser adaptados para diferentes tarefas após o treinamento. No contexto da medicina e da saúde, destacam-se aplicações como: *suporte a diagnósticos clínicos*, que melhora a precisão e a eficiência dos diagnósticos; *geração de relatórios clínicos*, que automatiza a criação de resumos e anotações de prontuários, reduzindo o tempo gasto pelos profissionais de saúde; *tradução de linguagem médica*, que facilita a comunicação com pacientes e o público em geral; *educação médica*, que promove o treinamento e a atualização de profissionais da saúde; *consulta e resposta médica*, que abrange triagem de consultas e respostas a pacientes; além de *atividades administrativas e de suporte*, como o atendimento ao público, que pode ser otimizado por meio de assistentes virtuais e chatbots.

6.2.7. Frameworks de desenvolvimento com LLMs

A construção de aplicações que utilizam grandes modelos de linguagem (LLMs) é significativamente facilitada pelo uso de frameworks de desenvolvimento especializados. Esses frameworks oferecem uma interface unificada para acessar diversos modelos e disponibilizam componentes pré-definidos para tarefas comuns, como indexação e armazenamento de operações anteriores. Ao combinar esses componentes, é possível criar aplicações complexas, como chatbots, assistentes virtuais e sistemas de busca, de maneira modular e flexível.

Dentre os frameworks disponíveis, o LangChain se destaca como um dos mais populares. Este framework de código aberto fornece uma interface padronizada para interagir com vários modelos de linguagem, como os desenvolvidos pela OpenAI, Google, Meta e outros modelos de código aberto. O LangChain permite a criação de aplicações de forma modular, facilitando a integração de diferentes componentes. As soluções de uso de LLMs no contexto da saúde apresentadas na seção 6.3 fazem uso deste framework.

No contexto do LangChain, um conceito fundamental é o de cadeia. Este con-ceito refere-se a uma sequência de operações ou componentes encadeados para realizar uma tarefa específica. Por exemplo, em um chatbot, é necessário encadear perguntas e respostas, preservando a memória da conversa. Outro exemplo é a sumarização de textos, que pode envolver a obtenção do texto, a geração de um prompt para instruir o modelo de LLM sobre o resumo e, finalmente, a produção da sumarização.

Dentre esses componentes destacam-se (LangChain 2023):

1. **Prompt Templates**: Modelos que facilitam a criação de prompts mais consistentes e eficazes.
2. **Chat History**: Mecanismos para armazenar e gerenciar o histórico de conversas, permitindo manter o contexto em interações contínuas.
3. **Document Loaders**: Ferramentas de importação e o carregamento de documentos em diferentes formatos, tornando-os acessíveis para o processamento nos modelos.
4. **Embedding Models**: Modelos que fornecem as representações vetoriais dos textos para operações com base em similaridade semântica.
5. **Vector Stores**: Armazenamentos que mantêm as representações vetoriais geradas pelos modelos de embedding, para recuperação eficiente de informações.

Uma lista completa desses componentes pode ser acessada em LangChain 2023 e muitos desses componentes são usados nas soluções aqui implementadas.

Exemplo: Uso direto do prompt. O prompt é a entrada que você fornece ao modelo de linguagem, como uma pergunta ou uma instrução. O LangChain permite que você utilize o prompt diretamente, mas sua interface cria uma camada de abstração que possibilita ser transparente o modelo empregado.

```
model_name = "llama/LLaMA-3-8B" # ou outro
llm = LlamaForCausalLM.from_pretrained(model_name)

prompt = "O que é HAS ou Hipertensão Arterial Sistêmica?"
response = llm(prompt)
print(response)
```

Código Fonte 6.6: Exemplo de uso direto de um prompt com o LangChain para obter uma resposta sobre Hipertensão Arterial Sistêmica.

Exemplo: Memória em Conversação. O mecanismo de memória permite que o modelo "lembre" de interações anteriores e mantenha a contextualização durante a conversa. O LangChain oferece diferentes tipos de memória, como memória de curto prazo e memória de longo prazo, para gerenciar e armazenar informações contextuais ao longo das interações.

```
from langchain.chains import ConversationChain
from langchain.memory import ConversationBufferMemory
from langchain.llms import OpenAI

model_name = "llama/LLaMA-3-8B" # ou outro
llm = LlamaForCausalLM.from_pretrained(model_name)

memory = ConversationBufferMemory()
conversation = ConversationChain(llm=llm, memory=memory)

conversation.run("O que é HAS ou Hipertensão Arterial Sistêmica?")
conversation.run("Quais são os tipos de tratamento?")
```

Código Fonte 6.7: Exemplo de uso do mecanismo de memória no LangChain para manter o contexto em uma conversação. O modelo responde a perguntas subsequentes com base na informação fornecida anteriormente.

O mecanismo de memória permite que o modelo mantenha a contextualização ao longo de uma conversação. Na segunda pergunta, “Quais são os tipos de tratamento?” o modelo já entende que se refere a tratamentos para a Hipertensão Arterial Sistêmica (HAS), devido ao contexto fornecido anteriormente.

Exemplo: Adição de Contexto. O contexto é uma técnica de ajuste fino que fornece informações adicionais ao modelo, permitindo a geração de respostas mais específicas e relevantes. O contexto pode incluir documentos diversos, como artigos, páginas, conjuntos de perguntas e respostas, ou relatórios. Todos esses documentos são traduzidos em texto adicional que é incorporado ao modelo para enriquecer suas respostas.

```
1
2 model_name = "llama/LLaMA-3-8B" # ou outro llm = LlamaForCausalLM.
      from_pretrained(model_name)
3
4 prompt_template = PromptTemplate( input_variables=["context", "question"],
      template="Você tem as seguintes informações: {context}\n\nCom base nessas
      informações, responda à seguinte pergunta: {question}" )
5
6 context = "Para HAS o tratamento não-medicamentoso consiste em (...). Para o
      tratamento medicamentoso são utilizadas as principais classes de agentes
      anti-hipertensivos, como diuréticos tiazídicos e inibidores da ECA."
7
8 question = "Quais são os principais tratamentos medicamentosos para HAS?" prompt
       = prompt_template.format(context=context, question=question) response =
      llm(prompt) print(response)
```

Código Fonte 6.8: Exemplo de adição de contexto usando LangChain. O código mostra como fornecer informações adicionais ao modelo para gerar respostas mais precisas com base no contexto fornecido.

No exemplo apresentado no Código 6.8, o modelo é esperado responder de forma mais precisa à pergunta sobre tratamentos medicamentosos para HAS, graças à informação contextual previamente fornecida.

6.3. Implementação de três diferentes soluções de chatbots inteligentes

A seguir apresentam-se 3 alternativas de Implementações de ChatBot para dados de Sáude (ver Figura 6.7). As três soluções empregam o modelo geral Meta-LLaMA-3-8B como base. Esse modelo pode ser substituído por outros modelos gerais como o GPT-4, BERT apenas fazendo-se uso da camada de abstração fornecida pelo LangChain.

1. **LLaMA**. O Modelo I emprega o modelo LLaMA diretamente, sem alterações. É uma abordagem simples, mas que pode ser útil para uma série de questões de saúde mais simples;

2. **LLaMA RAG**. O Modelo II, faz um ajuste fino integrando dados de contexto ao modelo base. Emprega-se para isso um grande documento o documento *Protocolo Clínico e Diretrizes Terapêuticas para Profilaxia Pré-Exposição (PrEP) de Risco à Infecção pelo HIV* (Saúde 2017) para obterem-se respostas mais acertivas sobre a profilaxia em casos de risco de infecção pelo HIV.

3. **LLaMA Customizado**. O Modelo III aplica um ajuste fino fazendo um sobre-treino do modelo original com um extenso conjunto de dados da Wikipedia sobre saúde. Neste ajuste foi empregado o ajuste-adaptativo LoRA (LowRank Adapta-tion) através do torchturne (PyTorch 2024).

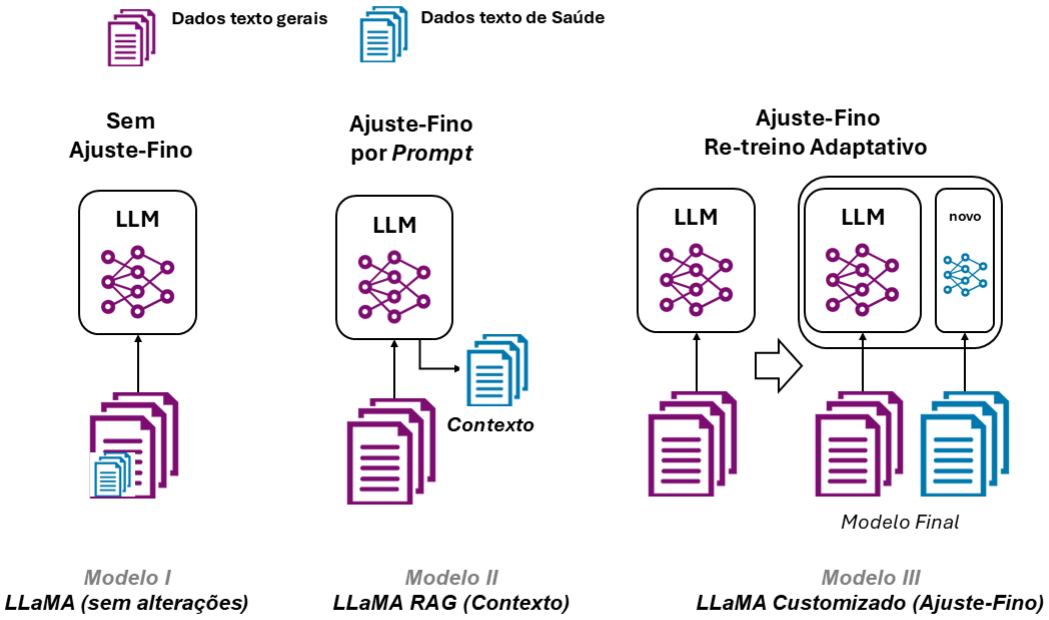

Figura 6.7: As três abordagens para chatbots inteligentes implementadas neste trabalho. Todas empregam o modelo Meta-LLaMA-3-8B AI@Meta 2024 como base. O Modelo I emprega o modelo LLaMA diretamente, sem alterações; O Modelo II, LLaMA RAG, faz um ajuste fino integrando dados de contexto ao modelo base; O Modelo III, LLaMA Customizado, aplica um ajuste- fino fazendo um sobre-treino do modelo original.

Essas três alternativas são integradas em um único aplicativo, o qual permite a seleção entre os diferentes modelos disponíveis. O código-fonte e os documentos relacionados podem ser acessados em [GitHub LLM], juntamente com materiais complementares. A solução utiliza o modelo Meta-LLaMA-3-8B (AI@Meta 2024) e o framework LangChain, além de destacar o uso do HuggingFace Instruct Embeddings para a vetorização dos documentos e da biblioteca FAISS (Facebook AI Similarity Search) para o armazenamento vetorial. Ambos são integrados através do LangChain (LangChain 2023). Para a interface do usuário, é empregado o Streamlit (Richards 2021), conhecido por sua simplicidade e facilidade de uso.

Embora a solução ofereça um protótipo funcional de chatbot para questões de saúde em três cenários distintos de uso dos LLMs, é crucial observar que o propósito é essencialmente conceitual. O protótipo está projetado para ser adaptado e refinado, mas não tem a intenção de servir como uma solução final, definitiva e otimizada para o complexo problema do uso de chatbots inteligentes na área da saúde.

6.3.1. Modelo I - LLaMA

O Modelo I - LLaMA, sem quaisquer ajustes, é a primeira alternativa de implementação de Chatbot. Apesar de não serem modelos especializados para responder a questões de saúde, os modelos de linguagem geral, como o LLaMA-3-8B (AI@Meta 2024), tendo sido treinados com um grande conjunto de dados, podem ser aplicados diretamente no caso de questões mais simples. O LLaMA-3-8B foi pré-treinado com mais de 15 trilhões de tokens de dados de fontes públicas e ajustado com conjuntos de dados de instruções e mais de 10 milhões de exemplos anotados por humanos. A versão menor, com 8 bilhões de parâmetros, foi escolhida para este protótipo por ser mais simples e mais adequada para os nossos propósitos. Ao contrário de modelos como o GPT-4, o LLaMA é um modelo aberto, disponível para download e ajuste sem custo, tornando-o acessível para experimentações e implementações.

```python

from transformers import LlamaForCausalLM, LlamaTokenizer

# Carregar o modelo e o tokenizer
model_name = "llama/LLaMA-3-8B"  # Substitua pelo nome correto do modelo
tokenizer = LlamaTokenizer.from_pretrained(model_name)
model = LlamaForCausalLM.from_pretrained(model_name)

def generate_text(prompt):
    # Tokenizar o texto de entrada
    inputs = tokenizer(prompt, return_tensors='pt')

    # Gerar uma resposta
    outputs = model.generate(
        inputs['input_ids'],
        max_length=100,
        num_return_sequences=1,
        no_repeat_ngram_size=2,
        temperature=0.7,
        top_p=0.9
    )

    # Decodificar a resposta
    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return response


# Exemplo de uso
prompt = "O que é HAS?"
response = generate_text(prompt)
print(response)
```

Código Fonte 6.9: Exemplo de código que demonstra como utilizar o modelo LLaMA-3-8B para gerar respostas a partir de um prompt. O código inclui a carga do modelo e do tokenizer, a geração de uma resposta com base em um contexto específico e a decodificação da resposta gerada.

Para ilustrar a aplicação direta do Meta-LLaMA-3-8B, o Código 6.9 fornece um exemplo de parte da solução. Inicialmente, é necessário carregar o modelo pré-treinado e o tokenizador adequado. No caso, utiliza-se o LlamaTokenizer da biblioteca Transformers, especificamente desenvolvido para a família de modelos LLaMA. Além disso o LLaMA é um modelo multilingue e, portanto, com suporte ao nosso idioma.

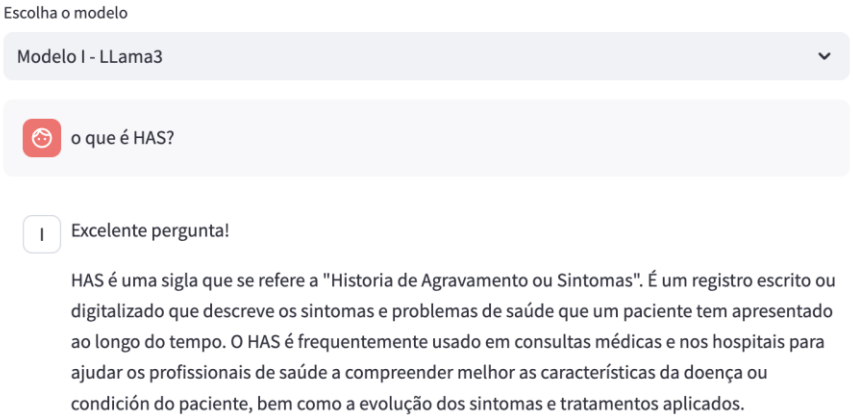

Figura 6.8: Resultado do prompt ao Modelo I – LLaMA.

Como o Modelo I - LLaMA foi utilizado sem ajustes adicionais e sem acesso a contexto externo, os resultados obtidos podem não ser suficientemente precisos. No exemplo apresentado, ao tentar responder a perguntas sobre "HAS", o modelo retorna "História de Agravamento ou Sintomas" em vez de "Hipertensão Arterial Sistêmica" como esperado (Figura 6.8). Esse resultado ilustra as limitações de se usar um modelo geral sem ajustes específicos para o contexto. Enquanto a abordagem pode ser adequada para questões simples e gerais, como "O que é um vírus?", perguntas mais complexas frequentemente exigem a inclusão de contexto adicional ou ajustes mais refinados do modelo para melhorar a precisão das respostas.

6.3.2. Modelo II - LLaMA RAG

Para melhorar a qualidade das respostas, uma abordagem mais elaborada é a incorporação de contexto externo por meio de Retrieval-Augmented Generation (RAG). Esse método envolve a adição de uma fonte de dados externa ao modelo, como documentos em formatos variados (.PDF, .CSV, .TXT). A inclusão de dados externos permite que o modelo acesse informações atualizadas que não estavam presentes no seu treinamento inicial, transformando-o potencialmente em um especialista em uma área específica.

Após obter os dados desejados, o processo inclui a vetorização e indexação desses dados em um banco de dados vetorial, utilizando a biblioteca FAISS como solução de armazenamento vetorial. Esse banco de dados permite ao modelo recuperar informações relevantes de maneira eficiente. As informações vetorizadas são então integradas ao modelo base por meio de uma interface que adapta o modelo para utilizar os dados externos durante a geração de respostas. Assim, ao receber um prompt, o modelo busca informações semelhantes e as utiliza como contexto para melhorar a precisão e a relevância das respostas fornecidas.

Para este propósito, foi utilizado o documento *Protocolo Clínico e Diretrizes Tera-pêuticas para Profilaxia Pré-Exposição (PrEP) de Risco à Infecção pelo HIV* ((Saúde 2017)). A inclusão desse contexto resultou em uma melhoria considerável

no desempenho do modelo, refletindo a eficácia da abordagem RAG em cenários mais especializados.

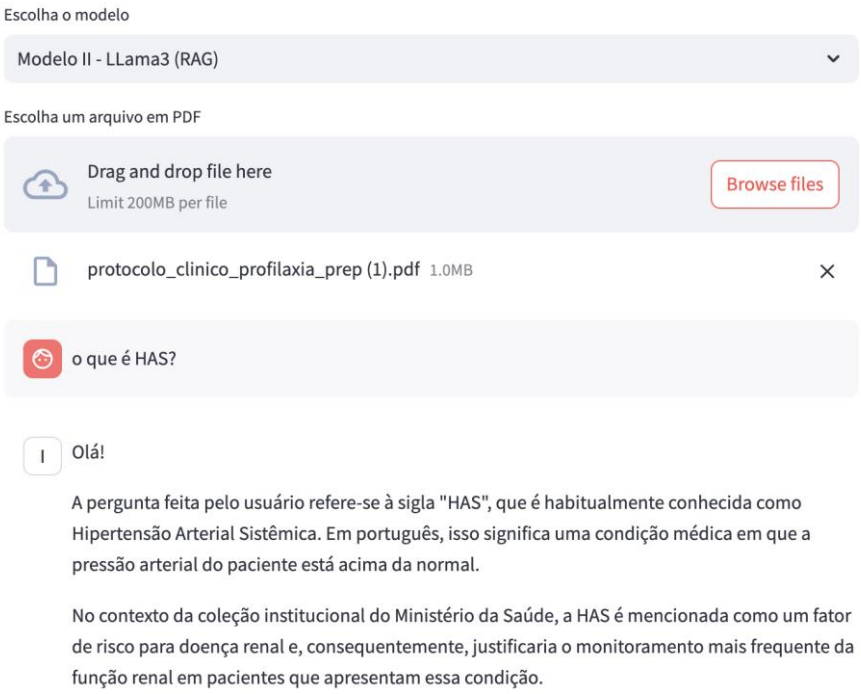

Figura 6.9: Resultado do prompt ao Modelo II - LLaMA RAG

6.3.3. Modelo III – LLaMA Customizado

Um aprimoramento adicional pode ser obtido com o ajuste fino do modelo utilizando um conjunto de dados de sobre-treinamento. Neste caso, foi empregado um extenso conjunto de dados coletado da Wikipedia sobre temas de saúde, acessível em [GitHub LLM]. A seleção dos documentos foi baseada em palavras-chave como Coronavírus, HIV, Câncer e Diabetes, e pode ser expandida para incluir outros termos relevantes (o código de extração está disponível em [GitHub LLM]).

Para o ajuste fino, foi utilizada a biblioteca Torchtune (PyTorch 2024). Torchtuneé uma biblioteca de otimização de hiperparâmetros para modelos PyTorch, oferecendo uma interface intuitiva para a configuração e ajuste dos modelos LLM. Ela simplifica o processo de ajuste ao automatizar a configuração de dados, o modelo e os loops de treinamento, reduzindo a necessidade de codificação manual. A abordagem empregada inclui o ajuste adaptativo LoRA (Low-Rank Adaptation), que melhora a eficiência do ajuste ao adicionar camadas novas de elementos sem alterar os pesos do modelo original.

O processo de ajuste fino pode ser realizado através de scripts que automatizam o treinamento, evitando a necessidade de codificação explícita em frameworks como Py-Torch ou TensorFlow. Embora essa abordagem ofereça menos flexibilidade para o programador, ela otimiza os parâmetros e proporciona bons resultados com um esforço redu-zido. Os principais passos para esse ajuste são descritos a seguir, e o script completo estádisponível em [GitHub LLM].

O dataset utilizado para o fine-tuning do modelo contém dados da Wikipedia em português sobre os seguintes temas: HIV, PrEP, AIDS e Profilaxia. Além dessas páginas, o dataset também inclui o PDF intitulado Protocolo Clínico e Diretrizes Terapêuticas para Profilaxia Pré-Exposição (PrEP) de Risco à Infecção pelo HIV. Este conjunto de dados foi projetado para permitir um treinamento especializado do modelo.

A. Preparação e Configuração do Ambiente. Envolve a instalação dos softwares necessários como o PyTorch, PyTorch Lightning, TorchTune e bitsandbytes, a carga do modelo e dos dados.

```
pip install torchtune bitsandbytes

tune download meta-llama/Meta-LLaMA-3-8B --output-dir /tmp/Meta-LLaMA-3-8B --hf-
    token <Your_TOKEN>

cp /path/to/torchtune/configs/llama3/8B_full_single_device.yaml /your/project/
    directory/
```

Código Fonte 6.10: Torchtune. Instalação e download do modelo.

```

# Configuração do Tokenizer
tokenizer:
  _component_: torchtune.models.llama3.llama3_tokenizer
  path: /tmp/Meta-LLaMA-3-8B/original/tokenizer.model

# Dataset
dataset:
  _component_: torchtune.datasets.alpaca_dataset
  train_on_input: True
seed: null
shuffle: True

# Modelo original
_component_: torchtune.models.llama3.llama3_8b

# Checkpoint
checkpointer:
  _component_: torchtune.utils.FullModelMetaCheckpointer
  checkpoint_dir: /tmp/Meta-LLaMA-3-8B/original/
  checkpoint_files: [
    consolidated.00.pth
  ]
  recipe_checkpoint: null
  output_dir: /tmp/Meta-LLaMA-3-8B/
  model_type: LLAMA3
resume_from_checkpoint: False

# Configuração do Treinamento
batch_size: 2
epochs: 3
optimizer:
  _component_: bitsandbytes.optim.AdamW8bit
  lr: 2e-5
loss:
  _component_: torch.nn.CrossEntropyLoss
max_steps_per_epoch: null
gradient_accumulation_steps: 1
optimizer_in_bwd: True
compile: False

# GPU?
device: cuda
```

Código Fonte 6.11: Torchtune. Principais configurações do treinamento do modelo.

Alguns modelos como o LLaMA requerem um <Your_TOKEN> obtido no site https://huggingface.co/ ou no site da Meta para o download. Um ar-quivo de

configuração padrão .yaml é fornecido ele pode ser editado para ajuste dos parâmetros do treinamento como o caminho do tokenizer e do dataset de dados, o modelo e as configurações de treinamento. Os principais elementos configurados encontram-se a seguir no 6.11.

O checkpoint especifica os parâmetros para salvar e carregar checkpoints do modelo durante o treinamento, que pode durar várias horas ou dias, dependendo dos dados e dos recursos de processamento. A configuração do treinamento abrange aspectos como métricas de eficiência, critérios de parada e otimizador, todos impactando o resultado e o tempo de processamento.

B. Execução do ajuste fino. O ajuste fino pode ser executado através do comando:

```
1
2 tune run full_finetune_single_device --config llama3/8B_full_single_device
```

Código Fonte 6.12: Torchtune. Principais configurações do treinamento do modelo.

Após a conclusão do ajuste, um novo arquivo de modelo é gerado e pode ser carregado pelo LangChain na aplicação. O resultado da mesma pergunta feita nas soluções anteriores é mostrado na Figura 6.10.

Figura 6.10: Resultado do prompt ao Modelo III - LLaMA Customizado

O resultado aqui não parece superior ao da solução anterior com contexto, o que ajuda a entender o desafio de se fazer ajuste de modelos como este. Em geral são necessários muito mais dados para esse tipo de treinamento, e uma avaliação de vários treinamentos com diferentes parâmetros pode ser necessária. Além disso, lembre-se que estamos fazendo emprego do modelo mínimo do LLaMA 3 (8B). Tendo por objetivo fornecer apenas um protótipo, ou uma prova de conceito das soluções, esse refinamento não será considerado aqui.

6.4. Conclusão

Os chatbots têm demonstrado um potencial significativo para transformar a forma como as informações médicas e de saúde são acessadas e processadas. Este capítulo examinou a aplicação de grandes modelos de linguagem (LLMs) na criação de chatbots inteligentes voltados para o setor da saúde, enfatizando a importância da personalização e do uso de dados próprios e modelos abertos.

Foi evidenciado que a personalização dos modelos de linguagem pode melhorar substancialmente a precisão e a relevância das respostas fornecidas pelos chatbots. A análise das três abordagens distintas — a utilização de um modelo genérico, a aplicação de Geração Aumentada de Recuperação (RAG) com dados externos e o ajuste fino com conjuntos de dados especializados — revelou como cada técnica contribui para aprimorar o desempenho dos chatbots na área da saúde. A abordagem RAG, ao integrar dados atualizados e relevantes, e o ajuste fino, ao adaptar os modelos a contextos específicos, evidenciam a evolução dos chatbots de soluções genéricas para sistemas mais refinados e eficazes.

Este estudo teve como objetivo fornecer uma prova de conceito das soluções e uma base para novos desenvolvimentos, sem ter a preocupação, portanto, com a eficácia e precisão dos resultados dos modelos. Isso exigiria modelos de linguagem maiores, muito maiores conjuntos de dados, mais recursos e horas de processamento, e complexas técnicas de ajuste de parâmetros dos modelos, comprometendo o fácil acesso, compreensão e uso direto das implementações pelo leitor.

É importante ressaltar ainda que a aplicação clínica dessas tecnologias requer uma colaboração estreita entre pesquisadores da computação e profissionais da saúde. A validação clínica e a implementação em ambientes reais requerem uma avaliação cuidadosa por especialistas médicos e a consideração de fatores éticos, regulatórios e de privacidade específicos.

Referências

[AI@Meta 2024] AI@Meta (2024). Llama 3 model card.

[Devlin et al. 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv pre-print arXiv:1810.04805*.

[GitHub LLM] Sato, F. M., Silva, J. E., Oliveira, R. de. Chabots Inteligentes na Saúde: Implementações com Modelos Abertos e Dados Próprios. Disponível em: https://github.com/Health-LLM-ChatBot/livro-llm

[Hochreiter and Schmidhuber 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[LangChain 2023] LangChain (2023). Langchain: Conceptual guide. Acesso em 27 de agosto de 2024.

[Mielke et al. 2021] Mielke, S. J., Alyafeai, Z., Salesky, E., Raffel, C., Dey, M., Gallé, M., Raja, A., Si, C., Lee, W. Y., Sagot, B., et al. (2021). Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*.

[Naveed et al. 2023] Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., and Mian, A. (2023). A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

[PyTorch 2024] PyTorch (2024). Welcome to the torchtune documentation. Acesso em 27 de agosto de 2024.

[Richards 2021] Richards, T. (2021). *Getting Started with Streamlit for Data Science: Create and deploy Streamlit web applications from scratch in Python*. Packt Publishing Ltd.

[Saúde 2017] Saúde, M. (2017). Protocolo clínico e diretrizes terapêuticas para profilaxia pré-exposição (prep) de risco à infecção pelo hiv. ISBN 978-65-5993-280-1. Acesso em 27 de agosto de 2024.

[Sutskever et al. 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

[Vaswani et al. 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Łukasz Kaiser, and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008.

[Wu et al. 2023] Wu, S., Fei, H., Qu, L., Ji, W., and Chua, T.-S. (2023). Next-gpt: Any-to-any multimodal llm. *arXiv preprint arXiv:2309.05519*.

[Zhou et al. 2023] Zhou, H., Gu, B., Zou, X., Li, Y., Chen, S. S., Zhou, P., Liu, J., Hua, Y., Mao, C., Wu, X., et al. (2023). A survey of large language models in medicine: Progress, application, and challenge. *arXiv preprint arXiv:2311.05112*.