Providing Task Execution Capabilities in LLM-Based Conversational Assistants

Nickolas Anselmo Carneiro Mororo
Universidade de Fortaleza - UNIFOR
Fortaleza, Ceará, Brasil
nickolasmororo@edu.unifor.br

Jorge Luiz Araújo
Universidade de Fortaleza - UNIFOR
Fortaleza, Ceará, Brasil
jorgearaujo@unifor.br

Rafael Bomfim
Universidade de Fortaleza - UNIFOR
Fortaleza, Ceará, Brasil
bomfim@unifor.br

Vasco Furtado
Universidade de Fortaleza - UNIFOR
Fortaleza, Ceará, Brasil
vasco@unifor.br

Abstract

Context: Conversational assistants (CAs) powered by Large Language Models (LLMs) excel in generating coherent responses, but struggle with task execution requiring long-term memory or temporal awareness. This research addresses such limitations, particularly in healthcare applications where task continuity is crucial.

Problem: Existing CAs often fail to meet user expectations in scenarios requiring long-term task management, such as scheduling reminders or monitoring chronic health conditions. This lack of temporal reasoning and memory continuity undermines user trust and engagement.

Proposed Solution: We propose a multi-agent system integrating LLMs with structured data processing, that enables CAs to recognize, manage, and execute user requests, such as scheduling reminders or tracking tasks over time. The solution bridges the gap between conversational fluency and actionable outcomes.

IT Theory: The study draws on theories of Human-Computer Interaction (HCI) and Temporal Awareness in Information Technology for allowing structured task execution.

Method: The research combines proof-of-concept implementation with healthcare case studies. Two experimental phases evaluated the solution: a pilot test with 35 users and an expanded trial with 437 participants focusing on chronic disease management.

Summarization of Results: Results demonstrated an improvement in task fulfillment and user engagement. The assistant successfully addressed 82% of task requests in simulations, with reduced user frustration and longer interactions during real-world trials.

Contributions and Impact on IT: The study enhances the capabilities of conversational assistants by introducing temporal reasoning and structured task execution, making them more effective in managing real-world tasks. These advancements improve the applicability of CAs, particularly in domains like healthcare, where accurate and timely task management is critical.

1 Introdução

O desenvolvimento de assistentes conversacionais (ACs), especialmente aqueles baseados em Amplos Modelos de Linguagem (*Large Language Models* ou LLMs), como o ChatGPT [10], tem o potencial de transformar aplicações de tecnologia da informação. Diferentemente dos chatbots tradicionais, que dependem de interações previsíveis e respostas programadas, gerando insatisfação e baixo engajamento dos usuários [3, 5, 6, 8], os ACs baseados em LLMs oferecem interações mais naturais e adaptáveis, aproximando-se de uma comunicação humana.

Essa evolução ocorre devido ao funcionamento dos LLMs, que utilizam arquiteturas de redes neurais profundas compostas por bilhões de parâmetros. Esses modelos são treinados em grandes corpora de texto, abrangendo vários domínios e estilos linguísticos. Durante o treinamento, técnicas como *unsupervised learning* e *self-supervised learning* permitem aos LLMs capturar padrões da linguagem, como relações semânticas, estruturas sintáticas e nuances contextuais. Ao receber uma entrada do usuário (*prompt*), o modelo processa a solicitação utilizando mecanismos como o *attention mechanism* e arquiteturas de transformadores (*transformer-based models*), que priorizam palavras e frases relevantes no contexto apresentado. A resposta gerada reflete a capacidade do modelo de prever a sequência mais apropriada de palavras, garantindo coerência e adequação ao estilo da conversa.

Mesmo com esse funcionamento, os LLMs enfrentam desafios. Um dos mais críticos é a limitação em relação à memória de longo prazo. Esses modelos são projetados para interações temporalmente limitadas, com capacidade restrita de vincular informações entre sessões. Ao término de uma interação, o modelo "esquece" os dados processados, inviabilizando a continuidade de tarefas que demandem acompanhamento histórico ou monitoramento prolongado. Esse fator restringe a aplicação de LLMs em cenários como gerenciamento de processos, suporte ao cliente e envio de lembretes personalizados, nos quais a continuidade é necessária.

Essa limitação é evidente ao observar o desempenho de agentes tradicionais, como Kokobot, Replika e Tess [4, 7, 9]. Embora ofereçam suporte emocional e alívio de estresse, esses agentes dependem de interações esporádicas e não conseguem manter um acompanhamento das necessidades dos usuários. Os LLMs, apesar de mais avançados no processamento de linguagem, enfrentam limitações semelhantes, pois suas respostas são baseadas em contextos pontuais, sem integração contínua de dados ou aprendizado adaptativo entre sessões.

Visando superar essas limitações, este estudo propõe uma arquitetura funcional que integra capacidades de assincronia e acompanhamento contínuo, incrementando a possibilidade dos assistentes em realizar tarefas que exigem conhecimento temporal. Diferentemente das abordagens existentes, essa arquitetura permite ao agente monitorar, registrar e retomar interações ao longo do tempo,

proporcionando suporte personalizado. Por meio da integração de funções especializadas, como o envio de lembretes e o ajuste dinâmico de interações baseadas em datas e horários específicos, o agente é capaz de gerenciar demandas de forma precisa e controlada. Essa abordagem representa um avanço na aplicabilidade de ACs em contextos que exigem monitoramento constante.

Ao incorporar funcionalidades de acompanhamento temporal, os LLMs podem transcender suas limitações estruturais e atender a um espectro mais amplo de aplicações, provendo a capacidade de realizar tarefas que requeiram domínio do tempo.

Essa nova funcionalidade foi implementada e avaliada na assistente conversacional de saúde, MarIA, no contexto de uma pesquisa em desenvolvimento em parceria com um grande provedor de plano de saúde nacional. Os experimentos avaliaram a eficácia do agente em realizar o acompanhamento contínuo e configurável de interações, demonstrando resultados promissores na capacidade de monitorar demandas ao longo do tempo e oferecer suporte personalizado com maior eficiência.

2 Contexto da Pesquisa

A MarIA é uma assistente virtual de Inteligência Artificial desenvolvida para promover o autocuidado em saúde, com foco em pacientes com doenças crônicas, especialmente a *Diabetes Mellitus*. Essa condição foi escolhida como caso de teste para a solução proposta neste artigo devido à sua complexidade e à necessidade de um acompanhamento contínuo. Diferente dos métodos tradicionais de acompanhamento — como telefonemas e consultas presenciais, que muitas vezes falham em incentivar a adesão ao tratamento devido à falta de personalização e limitações de escala [2] — MarIA utiliza Amplos Modelos de Linguagem (LLM) para ajustar sua comunicação de acordo com cada usuário. Essa personalização é realizada com base em características epidemiológicas, condições de saúde e hábitos específicos dos pacientes, permitindo um aconselhamento direcionado e individualizado.

O acompanhamento de pacientes com *Diabetes Mellitus* é essencial devido à natureza multifacetada da doença, que exige controle rigoroso da glicemia, adesão a tratamentos medicamentosos, ajustes na dieta e prática regular de atividades físicas. Falhas no controle desses fatores podem resultar em complicações graves, como neuropatia, insuficiência renal e doenças cardiovasculares. Nesse cenário, a capacidade de MarIA em oferecer monitoramento constante, lembretes oportunos e orientações específicas torna-se uma ferramenta valiosa para aumentar a adesão ao tratamento e prevenir complicações, atendendo às demandas desse perfil de pacientes de forma eficiente e escalável.

Construída com o protocolo da OpenAI, incluindo a API do *GPT Assistant* [10], MarIA foi projetada com um perfil funcional e comportamental claramente definido em suas *instructions*, que também estabelecem suas capacidades e limitações. A assistente interage com os usuários por meio do WhatsApp Web, utilizando a biblioteca Selenium e Python [1], o que facilita a comunicação contínua e acessível, especialmente em populações que já utilizam amplamente essa plataforma.

O experimento envolvendo MarIA foi realizado em duas fases e aprovado pelo comitê de ética, com registro na plataforma Brasil. A primeira fase contou com apenas 35 usuários convidados, dos quais 22 mulheres e 13 homens com idade média de 53 anos concordaram em participar. Essa etapa ocorreu entre 24 de novembro de 2023 e 22 de fevereiro de 2024, servindo como um teste inicial para avaliar a funcionalidade básica da assistente. Durante essa fase, observou-se que MarIA frequentemente fazia promessas aos usuários, como *"Vou lembrá-lo diariamente de ..."*, que não eram cumpridas. Essas promessas, relacionadas principalmente a sugestões de alimentação e lembretes, geraram expectativas não atendidas e evidenciaram uma lacuna na capacidade da assistente em compreender suas próprias limitações funcionais. Isso resultou em uma redução significativa no comprimento médio das mensagens enviadas pelos usuários, indicando menor engajamento e frustração com a ferramenta.

Na segunda fase, com as soluções propostas neste artigo implementadas, o experimento foi ampliado para incluir 437 pacientes de um programa de prevenção ao diabetes de um plano de saúde nacional, com idade média de 54 anos, sendo 247 mulheres e 187 homens. Essa etapa teve início em agosto de 2024 e continua em andamento no presente momento. Durante essa fase, MarIA passou a contar com funcionalidades como acompanhamento contínuo, configuração de lembretes dinâmicos e mensagens personalizadas, que permitiram à assistente alinhar suas ações com as expectativas dos usuários. As 18.378 interações válidas registradas até agora e analisadas por um especialista em saúde pública e três pesquisadores, que consideraram apenas as mensagens com textos descritivos e respostas dos participantes.

Os resultados preliminares da segunda fase mostram que o problema das promessas não cumpridas foi significativamente reduzido, e o engajamento dos participantes aumentou. Mensagens mais longas e detalhadas foram observadas, indicando maior disposição dos usuários para interagir com a assistente e um fortalecimento da confiança na ferramenta. A avaliação qualitativa também reforçou esses resultados, com relatos positivos dos participantes, como: *"Gostei muito dos lembretes que você configurou para minhas refeições"*. Esses achados destacam a importância de alinhar as capacidades de assistentes virtuais com as necessidades e expectativas dos usuários, especialmente em um contexto de saúde, onde a confiabilidade é essencial.

3 Solução Proposta

3.1 Arquitetura

De uma maneira geral, a arquitetura apresentada na Figura 1 mostra, idealmente, as etapas de raciocínio projetadas para um AC, como a MarIA, para que demandas de ações, como pedido de lembretes, feitas pelos usuários possam ser executadas. Assume-se que a realização de demandas do usuário requer conhecimento de mundo não explícito no diálogo e um planejamento de ações para alcançar o objetivo que se criou com a demanda. Em termos gerais, durante uma conversa, ao receber uma demanda do usuário, o AC primeiramente reconhece o objetivo, ou seja, a tarefa que deve ser realizada. Essa tarefa passa a ser o objetivo do AC e servirá para ativar o conhecimento de mundo que o LLM já possui. Desta forma, o modelo poderá realizar um planejamento para preencher os requisitos identificados através da rede de conhecimento e realizar as etapas necessárias para chegar ao objetivo definido. De posse desses requisitos, AC busca no texto os dados que especificam esse requisito.

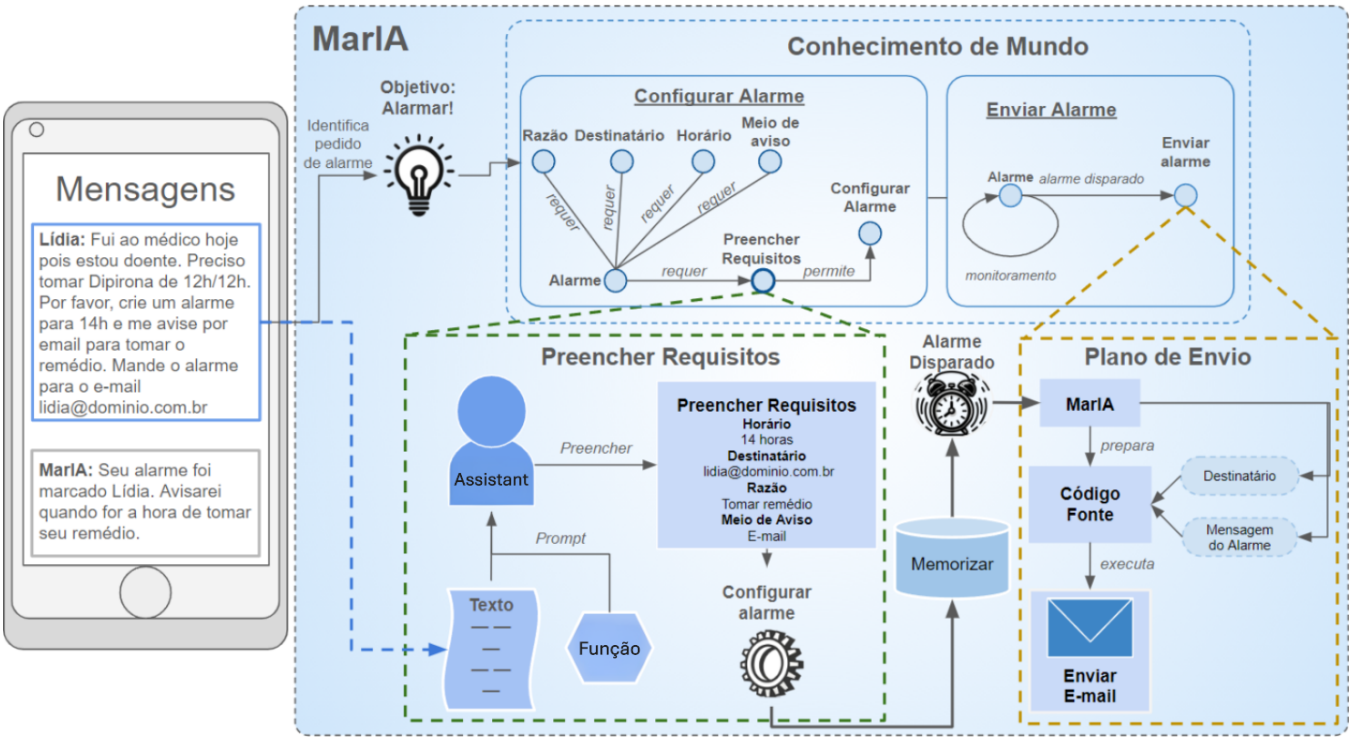

Figura 1: A arquitetura apresentada ilustra uma interação entre a MarIA e Lídia. Quando Lídia inicia uma conversa com a MarIA para solicitar um alarme, essa solicitação é transformada em um objetivo a ser alcançado pelo assistente. A partir desse objetivo, o conhecimento de mundo da MarIA é ativado, permitindo identificar as tarefas que precisam ser realizadas e os requisitos necessários para atingir o objetivo. A figura mostra a rede de conhecimento, a etapa de preenchimento dos dados necessários para configurar o alarme e o plano de envio, que garante que o alarme seja entregue ao paciente.

A figura apresenta um cenário de uma conversa entre a MarIA e uma usuária fictícia chamada Lídia. Nesta conversa, Lídia solicita à AC que marque um alarme às 14 horas, pois ela precisa tomar um remédio, e que mande este alarme por e-mail. A partir desta demanda, o objetivo "Alarmar"é criado. Este objetivo será usado para ativar a rede de conhecimento interna da AC, desta forma, mais informações envolvendo o tema poderão ser usadas para definir quais serão as próximas ações da assistente. Neste exemplo, duas partes da rede de conhecimento estão destacadas, referentes às etapas de "Configurar Alarme"e "Enviar Alarme". A rede de conhecimento interna ajuda a AC a reconhecer quais são os requisitos necessários para a tarefa, como, por exemplo, para configurar um alarme é essencial saber qual o horário desejado.

Esses requisitos são extraídos diretamente da mensagem inicial enviada por Lídia. A extração será realizada pelo agente de Extração de Informações, através de um prompt. Após identificar o "Horário, Destinatário,Razão,"e "e-mail,"o alarme é configurado e registrado.

Com o alarme configurado, é possível seguir para a etapa de "Enviar Alarme". Quando o alarme é disparado, a rotina de "Envio do e-mail"é iniciada. AC executa o código-fonte que receberá os requisitos necessários para a criação e envio do e-mail; neste caso, serão passados para o código-fonte o destinatário da mensagem e uma mensagem personalizada. Por fim, ocorrerá a chamada a um servidor que permitirá o envio da mensagem, completando a demanda da paciente de ser alertada para tomar seu remédio.

A arquitetura proposta é projetada para ser escalável e adaptável a diferentes domínios além da saúde, como educação e suporte ao cliente. Sua capacidade de operar de forma independente do contexto específico permite que o modelo de conversação do agente seja ajustado conforme os critérios de cada aplicação, incluindo prioridade, reconhecimento de contexto e estratégia de resposta. O sistema é executado por meio de APIs, garantindo integração com diferentes plataformas e permitindo comunicação entre os módulos. Para garantir escalabilidade em grandes bases de usuários, a solução adota uma estrutura modular e distribuída, permitindo a alocação eficiente de recursos computacionais. No contexto educacional, por exemplo, o assistente pode gerenciar calendários acadêmicos, prazos e lembretes, oferecendo suporte a um grande número de usuários.

3.2 Extração de informações

O framework *GPT Assistants*, da OpenAI [10], foi utilizado no desenvolvimento da solução devido à sua capacidade de integrar funções externas e executar tarefas conectadas a um *backend* em Python. Esse framework possibilita a criação de agentes baseados em LLMs, chamados *assistants*, que interagem com os usuários de forma adaptável e eficaz.

No *GPT Assistants*, a execução de funções ocorre por meio de dois componentes principais: o *frontend* e o *backend*. O *frontend* atua como interface gráfica, descrevendo as ações que o *assistant* deve realizar ou as informações que deve coletar durante as interações com o usuário. O *backend* é responsável pela execução das funções Python locais, processando as informações estruturadas no *frontend*. Por exemplo, na função de marcação de alarmes, o *assistant* identifica informações como data, hora e motivo na conversa, organiza esses dados no *frontend* e os envia ao banco de dados de alarmes no *backend*. Além disso, o agente temporal utiliza funções do módulo `datetime.datetime` no *backend* para consultar a hora atual, gerenciando intervalos de tempo de maneira autônoma.

O agente temporal conta com três funções principais no *frontend*, otimizadas para o gerenciamento eficiente de eventos e alarmes:

- **`hora_atual`**: Realiza consultas ao horário atual sem necessidade de parâmetros adicionais. A ativação ocorre de acordo com o contexto da conversa. Detalhes sobre seu funcionamento estão apresentados na Figura 2.
- **`monitoramento_temporal`**: Desenvolvida para acompanhar pacientes com condições de saúde que exigem monitoramento contínuo. Essa função utiliza inferências temporais para priorizar tarefas e programar acompanhamentos, capturando parâmetros como *subject*, *priority*, *request_timestamp* e *next_checkup*. Os dados são organizados em formato JSON e enviados ao *backend* para configuração de alarmes ou lembretes. A Figura 3 apresenta mais detalhes.
- **`informacoes_temporais`**: Captura dados essenciais, como hora, data e motivo, necessários para a configuração de alarmes e lembretes. As informações extraídas da conversa são estruturadas em JSON e enviadas ao *backend* para processamento. A Figura 4 provem detalhes adicionais.

Algoritmo 1: Definição da Função de Hora Atual

```
{"name": "hora_atual",
  "description": "Utilize esta função para obter a data e a hora
  ↪  atuais. Esta função é crucial para que o agente mantenha
  ↪  uma noção precisa do tempo presente, permitindo realizar
  ↪  cálculos temporais, configurar alarmes ou lembretes e
  ↪  acompanhar eventos de relevância temporal. Sempre que uma
  ↪  situação temporal surgir, o agente deve persistentemente
  ↪  chamar essa função para se situar corretamente no tempo.
  ↪  Em caso de dúvida, ou se houver qualquer incerteza sobre a
  ↪  cronologia de eventos, chame a função sem hesitação para
  ↪  garantir que as informações estejam sempre atualizadas.",
  "strict": true,
  "parameters": {
    "type": "object",
    "properties": {},
    "additionalProperties": false,
    "required": []
  }
}
```

Figura 2: Estrutura da função `hora_atual`, que fornece ao agente a data e hora atuais. Essa função é fundamental para que o agente mantenha uma noção precisa do tempo, permitindo cálculos temporais e o ajuste adequado de alarmes e lembretes, garantindo a atualização constante das informações temporais no contexto das interações.

A definição das funções no *frontend* é essencial para orientar o sistema sobre as informações específicas que devem ser monitoradas e extraídas. Essa abordagem permite ao agente priorizar

Algoritmo 2: Definição da Função de Informações Temporais

```
{"name": "informacoes_temporais",
  "description": "Utilize esta função para salvar lembretes
  ↪  solicitados pelo usuário, assegurando a captura de
  ↪  informações essenciais (data, hora e motivo). Se
  ↪  necessário, solicite informações adicionais para
  ↪  personalização, como repetição e detalhes de recorrência.
  ↪  Antes de proceder, confirme que o usuário deseja criar um
  ↪  alarme ou lembrete. Use o dia atual para cálculos e
  ↪  verificações, incluindo horários relativos como 'daqui a 5
  ↪  minutos'. Para alarmes recorrentes, colete informações
  ↪  detalhadas sobre a frequência da recorrência, bem como as
  ↪  datas de início e término.",
  "strict": true,
  "parameters": {
    "type": "object",
    "properties": {
      "hora": {
        "type": "string",
        "description": "A hora em que o alarme será ativado deve
        ↪  ser especificada no formato HH:mm (24 horas). Caso o
        ↪  horário seja fornecido de forma relativa, como
        ↪  'daqui a 5 minutos', o modelo deve ser capaz de
        ↪  interpretar essa informação, realizar os cálculos
        ↪  necessários para determinar a hora exata do alarme e
        ↪  ajustar automaticamente para o horário correto. Isso
        ↪  inclui interpretar comandos relativos ao tempo
        ↪  atual, como 'em uma hora' ou 'em 30 minutos', e
        ↪  calcular a hora exata em que o alarme deve ser
        ↪  ativado."
      },
      "data": {
        "type": "string",
        "description": "O dia em que o alarme será ativado, no
        ↪  formato YYYY-MM-DD. Se não especificado, o alarme
        ↪  será configurado para o dia atual."
      },
      "motivo": {
        "type": "string",
        "description": "A razão para o alarme. Por exemplo:
        ↪  'tomar remédio' ou 'medir a glicemia'. Deve ser
        ↪  claro e específico."
      }
    },
    "additionalProperties": false,
    "required": [
      "hora",
      "data",
      "motivo"
    ]
  }
}
```

Figura 3: Especificação da função `informacoes_temporais`, responsável por capturar informações essenciais para criação de lembretes, como data, hora e motivo. Esta função permite que o agente interprete comandos de tempo relativos e configure alarmes personalizados, garantindo que o usuário seja avisado de eventos importantes.

eventos, configurar alarmes e organizar dados relevantes de forma estruturada.

Como exemplo ilustrativo, pode-se considerar a interação com a usuária Lídia: *"Meu nome é Lídia, tenho 28 anos. Estou sentindo dor de cabeça constante há cerca de três dias. Também notei uma febre leve ontem à noite. Estou preocupada se poderia ser algo relacionado ao estresse ou se é algo mais sério."* Nesse contexto, a função `monitoramento_temporal` foi acionada para identificar e organizar informações temporais e de saúde extraídas da conversa. A Figura 5 demonstra como os dados capturados são estruturados em formato JSON, incluindo o assunto relatado ("Dor de cabeça constante e febre leve"), a prioridade atribuída ("alta"), o registro do *timestamp* da solicitação e o agendamento do próximo acompanhamento.

Após a captura e organização no *frontend*, as informações são enviadas ao *backend*, onde são processadas para instanciar um alarme.

Algoritmo 3: Definição da Função de Monitoramento Temporal

```
{"name": "monitoramento_temporal",
  "description": "Utilize esta função para monitorar assuntos
  ↪ relevantes com base em inferências temporais e configurar
  ↪ alarmes para eventos de alta prioridade. O agente deve
  ↪ estipular, com base em seu próprio conhecimento e
  ↪ avaliação, a prioridade ('priority') de cada assunto e a
  ↪ data do próximo acompanhamento ('next_checkup').",
  "strict": true,
  "parameters": {
    "type": "object",
    "properties": {
      "requests": {
        "type": "array",
        "items": {
          "type": "object",
          "properties": {
            "subject": {
              "type": "string",
              "description": "O assunto relevante a ser
              ↪ monitorado."
            },
            "priority": {
              "type": "string",
              "description": "Prioridade atribuída ao assunto,
              ↪ estipulada pelo agente com base em seu próprio
              ↪ conhecimento e avaliação."
            },
            "request_timestamp": {
              "type": "string",
              "description": "Data e hora em que o assunto foi
              ↪ registrado no formato 'YYYY-MM-DD HH:MM'."
            },
            "next_checkup": {
              "type": "string",
              "description": "Data estipulada pelo agente para o
              ↪ próximo acompanhamento do assunto no formato
              ↪ 'YYYY-MM-DD HH:MM'."
            }
          },
          "required": [
            "subject",
            "priority",
            "request_timestamp",
            "next_checkup"
          ],
          "additionalProperties": false
        }
      },
      "final_summary": {
        "type": "string",
        "description": "Resumo final com informações gerais
        ↪ sobre os acompanhamentos e alarmes configurados."
      }
    },
    "additionalProperties": false,
    "required": [
      "requests",
      "final_summary"
    ]}}
```

Figura 4: Definição da função `monitoramento_temporal`, utilizada para monitorar assuntos relevantes com base em inferências temporais e configurar alarmes para eventos de alta prioridade. A função permite que o agente estipule a prioridade (*priority*) e a data do próximo acompanhamento (*next_checkup*) com parâmetros estruturados para identificar o assunto monitorado, a prioridade, o horário da solicitação e o próximo acompanhamento.

Esse alarme contém o motivo fornecido pela usuária ("Dor de cabeça constante e febre leve"), o horário atual registrado na interação e a data do próximo checkup, estipulada pelo próprio agente com base em seu conhecimento de mundo sobre a relevância do motivo relatado. Essa integração entre *frontend* e *backend* possibilita o gerenciamento automatizado de eventos e tarefas, garantindo um acompanhamento contínuo e eficiente.

```
"requests": {
  "subject": "Dor de cabeça constante e febre leve",
  "priority": "alta",
  "request_timestamp": "2023-10-06 10:00",
  "next_checkup": "2023-10-08 10:00"
},
"final_summary": "Monitoramento de saúde: dor de cabeça constante e
↪ febre leve."
```

Figura 5: Informações extraídas pelo agente na interação com a usuária Lídia, organizadas em JSON: o agente capturou o assunto da queixa ("Dor de cabeça constante e febre leve"), definiu a prioridade como "alta", registrou o timestamp da solicitação e agendou o próximo acompanhamento.

3.3 Instanciação de Alarmes

O agente desenvolvido utiliza funções específicas para capturar informações e consultar a hora atual. Sempre que identifica uma possível solicitação de alarme na conversa, ele gera o objetivo de recuperar a hora atual para garantir que o alarme seja configurado corretamente. Essa consulta ajusta o horário fornecido pelo usuário ao contexto temporal real, enviando o horário correto ao *backend* para a configuração do temporizador, conforme ilustrado no processo de "Preencher Requisitos"na Figura 1.

Caso o horário solicitado já tenha passado, o agente detecta a inconsistência ao comparar o horário fornecido com a hora atual. Esse processo, baseado no objetivo de validação temporal, permite ao agente informar o usuário que o alarme é inválido e solicitar uma correção antes de prosseguir. Esse fluxo assegura que apenas horários válidos sejam configurados.

Após validar as informações, o agente organiza os dados capturados, como motivo e horário, em formato JSON e os envia ao *backend*. Nesse ponto, o agente define o objetivo de marcar o temporizador, configurando-o no *backend* para o horário especificado. O temporizador opera de forma assíncrona em uma thread separada e, ao atingir o horário programado, apenas notifica o agente de que o momento do alarme chegou, como descrito no plano de envio mostrado na Figura 1.

Quando o alarme é disparado, o agente gera um novo objetivo: recuperar o motivo associado ao horário do alarme. Para isso, utiliza o histórico da conversa, buscando a descrição fornecida pelo usuário no momento da solicitação. Assim, o agente compõe a notificação com o motivo correto e envia ao usuário uma mensagem, como "Olá, lembre-se do seu alarme programado agora com a descrição motivo."

Essa abordagem, orientada por objetivos, permite ao agente realizar múltiplas tarefas de maneira integrada, como recuperar a hora atual, configurar temporizadores, validar horários e buscar informações no histórico da conversa. Além disso, ela possibilita o gerenciamento simultâneo de múltiplos alarmes, garantindo que o usuário receba lembretes no momento correto e com a descrição apropriada, conforme ilustrado no fluxo completo da Figura 1. Essa arquitetura assegura uma interação eficiente e baseada em contextos definidos pelo próprio agente.

Para garantir a correta configuração de alarmes e lembretes, o assistente adota uma abordagem iterativa para lidar com entradas ambíguas. Sempre que um usuário solicita um lembrete sem

fornecer todas as informações necessárias, o agente solicita esclarecimentos antes de registrar o evento. Por exemplo, se o usuário disser apenas “Lembre-me de tomar remédio”, o sistema perguntará “Qual horário?” antes de concluir a configuração do lembrete.

Além disso, o grau de prioridade dos lembretes e acompanhamentos é estipulado diretamente pelo agente, fundamentado tanto no *Prompt* de instrução, que define as diretrizes operacionais do modelo, quanto no seu próprio conhecimento de mundo. Dessa forma, situações críticas são identificadas automaticamente e recebem um acompanhamento mais próximo, enquanto interações rotineiras são distribuídas em intervalos maiores.

Em casos de conflitos de agendamento, o assistente avalia a urgência dos eventos e sugere reagendamentos quando necessário, garantindo que demandas prioritárias não sejam comprometidas. Essa estratégia permite que o sistema gerencie múltiplos lembretes de maneira eficiente, evitando sobrecargas ou informações conflitantes.

3.4 Recorrência e Integração com Sistemas Externos

Durante os experimentos, a capacidade da MarIA, um agente conversacional de saúde para pacientes com diabetes mellitus, de configurar alarmes recorrentes mostrou-se essencial para apoiar atividades rotineiras. O agente analisava o contexto das conversas com os usuários para identificar padrões e momentos que demandavam lembretes regulares. Além de responder a solicitações diretas, fazia sugestões de alarmes com base nas interações, como no caso de um paciente que precisava de lembretes para medir a glicemia antes das refeições. Essa funcionalidade de sugerir alarmes recorrentes, no entanto, não havia sido prevista pela equipe de pesquisa e surgiu como uma "surpresa agradável"ao observar o comportamento do agente.

Em um desses casos, ao receber a informação sobre os horários das refeições, a MarIA programou alarmes para os momentos adequados ao longo do dia, configurando-os como recorrentes para os dias subsequentes. Essa automação eliminou a necessidade de reprogramações manuais, garantindo a regularidade das medições e incentivando o reengajamento periódico do paciente com o agente. A capacidade do agente de propor lembretes ajustados às necessidades individuais reforçou sua utilidade no apoio ao gerenciamento da rotina dos pacientes.

A configuração de alarmes recorrentes ampliou o alcance da MarIA, permitindo atender a demandas que exigem frequência e organização contínuas, como o controle glicêmico ou a administração de medicamentos. O agente ajustava os lembretes ao contexto extraído das conversas, identificando necessidades explícitas ou implícitas para propor lembretes relevantes e alinhados ao comportamento do usuário. Essa abordagem reduziu a carga de intervenções manuais e facilitou o acompanhamento das rotinas de saúde.

A capacidade de sugerir e configurar alarmes recorrentes com base nas interações consolidou a MarIA como uma ferramenta prática e adaptável no acompanhamento de pacientes com diabetes mellitus. Ao oferecer suporte proativo e personalizado, o agente promoveu a regularidade em tarefas críticas para o cuidado com a saúde, como a monitorização glicêmica, fortalecendo a adesão às rotinas recomendadas.

O acompanhamento recorrente proporcionado pela MarIA revelou-se fundamental para os pacientes, pois permitiu manter a regularidade no autocuidado, promover organização nas atividades de saúde e reforçar o contato contínuo com o agente. Essa funcionalidade destacou o potencial de sistemas baseados em IA no suporte ao gerenciamento de doenças crônicas, integrando inteligência contextual e suporte adaptável às rotinas de saúde dos usuários.

Durante a implementação do assistente, a integração com sistemas externos apresentou desafios relacionados à latência e à consistência dos dados. Para mitigar esses problemas, foram adotadas chamadas assíncronas, que reduziram os tempos de resposta e garantiram uma comunicação mais fluida entre os módulos do sistema. Além disso, um banco de dados foi incorporado para armazenar o histórico de interações, assegurando continuidade na personalização dos lembretes e alarmes.

A sincronização periódica foi implementada para garantir que as interações anteriores fossem consideradas na configuração de novos lembretes. No caso do WhatsApp, foi desenvolvida uma automação baseada em uma fila FIFO (First-In, First-Out), permitindo que as notificações fossem enviadas de maneira sequencial e evitando congestionamento de mensagens. Essas soluções aprimoraram a confiabilidade e a eficiência do sistema, garantindo que os usuários recebessem lembretes no momento correto e com informações consistentes.

4 Avaliação

A avaliação do desempenho do *assistant* na identificação e configuração de alarmes em cenários cotidianos foi conduzida utilizando uma metodologia baseada em sistemas multiagentes orientados por LLMs. O objetivo central era testar a capacidade do *assistant* de interpretar solicitações de alarmes, extrair informações relevantes e configurá-los corretamente em diferentes contextos.

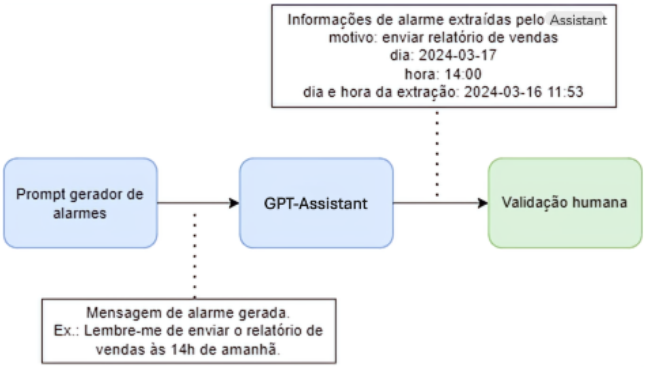

Figura 6: Fluxo de avaliação do agente: Uma requisição de alarme é gerada e disponibilizada para que o agente identifique se o texto se refere a uma requisição de alarme. Caso seja confirmado que se trata de uma requisição de alarme, o agente extrai o motivo, a data e o horário do alarme. Posteriormente, a validação humana é realizada com base nas informações extraídas.

O fluxo metodológico, ilustrado na Figura 6, envolve a criação de um dataset contendo 100 frases, das quais 80 apresentam pedidos de

alarmes e 20 são aleatórias, sem relação com a tarefa. Essas frases foram submetidas ao *assistant* para determinar se configuravam solicitações válidas e, em caso positivo, realizar a extração e registro de informações como motivo, data e horário do alarme. A validação dos resultados foi feita por meio da comparação com uma coleção dourada, elaborada por um analista humano e utilizada como referência.

> Comporte-se como uma pessoa que tem dificuldades em lembrar das atividades do dia a dia e utiliza um sistema de lembretes. Você tem acesso a esse sistema para ajudar a lembrar de várias situações cotidianas. Gere 100 ocorrências de lembretes com variação na forma de requisitar, incluindo detalhes explícitos sobre momentos (data e hora), períodos em minutos, horas, dias, semanas e meses à frente, além de lembretes recorrentes. Mantenha a diversidade em termos de estrutura e expressões, abrangendo aspectos sintáticos e semânticos variados para desafiar a detecção de aspectos temporais.

Figura 7: Prompt utilizado para gerar um conjunto de testes com lembretes variados. Esse corpo de testes foi criado para avaliar a acurácia do agente em identificar e processar informações temporais, incluindo lembretes únicos, recorrentes e em diferentes formatos contextuais.

O processo incluiu cinco iterações com as mesmas 100 frases, permitindo maior consistência na avaliação. Como resultado, o *assistant* demonstrou uma alta taxa de acerto na identificação e processamento dos alarmes, evidenciando sua capacidade de realizar a tarefa. A média de acurácia observada ao longo das iterações foi de 82%, com resultados positivos tanto em alarmes simples quanto em situações mais complexas, o que será detalhado em análises futuras.

Os valores apresentados na figura 8 refletem as métricas médias obtidas a partir de uma matriz de confusão que é a média dos resultados de cinco outras matrizes de confusão provenientes de uma avaliação. Isso indica que o modelo foi testado em cinco ocasiões distintas, e os resultados foram combinados para fornecer uma visão geral mais robusta do seu desempenho. Com uma acurácia média de 66%, precisão de 82%, recall de 71% e F1-Score de 76%, os dados sugerem que o modelo possui uma boa capacidade de prever corretamente os casos positivos, mas ainda há espaço para melhorias, especialmente na redução de falsos negativos e no aumento da acurácia geral.

Apesar do desempenho geral satisfatório, o *assistant* enfrentou desafios em contextos temporais específicos. Por exemplo, ele apresentou dificuldades em interpretar solicitações relacionadas a dias específicos da semana, como "próxima sexta-feira", ou eventos baseados em feriados. Essas falhas foram identificadas em frases como: "Preciso de um lembrete semanal para regar as plantas todas as quartas-feiras" e "Lembre-me de colocar um lembrete para o próximo feriado".

Por outro lado, o *assistant* obteve sucesso em alarmes de contexto direto, como horários fixos. Um destaque foi sua precisão ao programar alarmes para horários como o almoço e o término do expediente, que foram configurados corretamente. Exemplos desses casos incluem as solicitações: "Por favor, lembre-me de fazer compras no supermercado na hora do almoço" e "Peça para me lembrar de comprar flores na saída do trabalho."

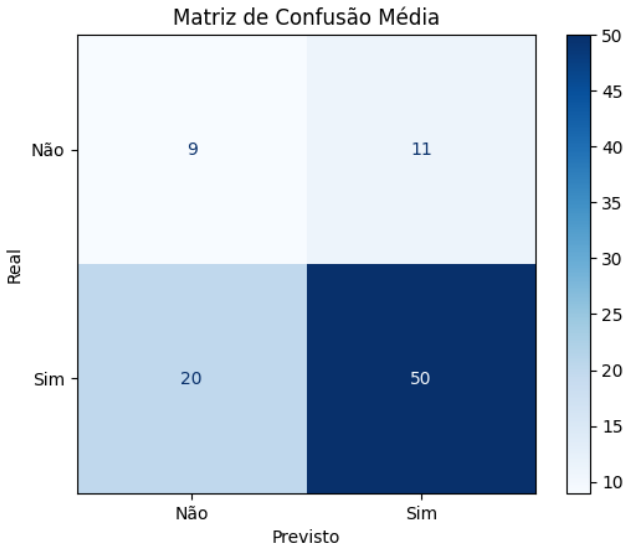

Figura 8: Matriz de confusão média para detecção de alarmes, consolidando os resultados de cinco testes. Os quadrantes representam os diferentes cenários: acertos ao detectar alarmes corretamente (era alarme e marcou), acertos ao não detectar alarmes inexistentes (não era alarme e não marcou), erros por falha na detecção de alarmes presentes (era alarme e não marcou) e erros ao identificar alarmes que não existiam (não era alarme e marcou).

Esses resultados confirmam que o *assistant* possui uma alta capacidade de realizar a tarefa proposta, especialmente em cenários de baixa complexidade. As análises realizadas fornecem insights valiosos para refinar o desempenho do *assistant* em contextos mais desafiadores, consolidando sua utilidade em aplicações cotidianas e destacando seu potencial para atender a solicitações complexas com maior precisão no futuro.

5 Conclusão

Este artigo apresentou uma abordagem para o desenvolvimento de assistentes conversacionais (ACs) baseados em Amplos Modelos de Linguagem (LLMs), abordando limitações na realização de tarefas práticas e no suporte contínuo. A solução proposta integra um sistema multiagente que combina processamento de linguagem natural com capacidades temporais, permitindo que os ACs atendam a demandas dos usuários, como a configuração de alarmes e lembretes. Essa integração melhora a funcionalidade e a utilidade dos assistentes em aplicações práticas.

Os resultados dos experimentos demonstraram o desempenho do sistema em contextos simulados e reais. A avaliação inicial indicou que a arquitetura multiagente permite uma interação eficiente, com boa taxa de acerto na interpretação de solicitações. Essa eficácia foi observada principalmente em cenários de menor complexidade, como lembretes simples, embora desafios permaneçam em contextos com referências temporais vagas ou altamente específicos. Esses resultados destacam tanto os avanços alcançados quanto áreas que precisam de aprimoramento.

A aplicação prática do sistema na assistente MarIA mostrou sua utilidade no gerenciamento de condições de saúde crônicas, como diabetes mellitus. A capacidade do agente de configurar lembretes recorrentes e monitorar necessidades dos pacientes resultou em interações mais longas e maior engajamento. Os relatos dos participantes reforçam a importância de integrar capacidades temporais aos ACs, especialmente em cenários que exigem suporte contínuo.

Apesar dos avanços, o artigo identificou limitações que orientam os próximos passos. A interpretação de referências temporais vagas e a desambiguação em contextos variados são áreas que demandam atenção. O sistema também pode ser expandido para suportar tarefas mais complexas, como o planejamento de longo prazo e a adaptação a novas demandas dos usuários. A inclusão de técnicas de aprendizado contínuo e modelos especializados pode contribuir para superar essas limitações.

Como parte do desenvolvimento futuro, propõe-se a realização de testes em outros contextos além da saúde, avaliando o desempenho da solução em ambientes educacionais, empresariais e domésticos. Esse direcionamento permitirá explorar a aplicabilidade da arquitetura multiagente em diferentes cenários, ampliando sua utilidade.

Este artigo contribuiu para o avanço dos assistentes conversacionais na realização de tarefas práticas e no suporte contínuo. A integração de funcionalidades temporais e a abordagem orientada por objetivos tornam os ACs mais eficientes. Ao superar as limitações atuais e expandir as aplicações, a solução apresentada tem o potencial de melhorar a interação entre usuários e sistemas de inteligência artificial.

Referências

[1] Baiju Muthukadan. 2024. Selenium Python Bindings. https://selenium-python.readthedocs.io/.

[2] Serena Barello, Guendalina Graffigna, and Elena Vegni. 2012. Patient engagement as an emerging challenge for healthcare services: Mapping the literature. *Nurs Res Pract* 2012 (2012). https://doi.org/10.1155/2012/905934

[3] Ghazala Bilquise, Samar Ibrahim, and Khaled Shaalan. 2022. Emotionally Intelligent Chatbots: A Systematic Literature Review. *Human Behavior and Emerging Technologies* (2022). https://doi.org/10.1155/2022/9601630

[4] G Dosovitsky, BS Pineda, NC Jacobson, C Chang, M Escoredo, and EL Bunge. 2020. Artificial intelligence chatbot for depression: descriptive study of usage. *JMIR Formative Research* 4 (2020), e17065. https://doi.org/10.2196/17065

[5] Meghana Gudala, Sunitha Mogalla, Mandi Lyons, Padmavathy Ramaswamy, Mary Ellen Ross, and Kirk Roberts. 2022. Benefits Of, Barriers To, and Needs for an Artificial Intelligence–Powered Medication Information Voice Chatbot for Older Adults: Interview Study With Geriatrics Experts. *Jmir Aging* (2022). https://doi.org/10.2196/32169

[6] Magdalena Görtz, Kilian Baumgärtner, T Schmid, Marc Muschko, Philipp Woessner, Axel Gerlach, Michael Byczkowski, Holger Sültmann, Stefan Duensing, and Markus Hohenfellner. 2023. An Artificial Intelligence-Based Chatbot for Prostate Cancer Education: Design and Patient Evaluation Study. *Digital Health* (2023). https://doi.org/10.1177/20552076231173304

[7] Kien Hoa Ly, Ann-Marie Ly, and Gerhard Andersson. 2017. A fully automated conversational agent for promoting mental well-being: a pilot RCT using mixed methods. *Internet Interventions* 10 (2017), 39–46. https://doi.org/10.1016/j.invent.2017.10.002

[8] Abdollah Mahdavi, Masoud Amanzadeh, Mahnaz Hamedan, and Roya Naemi. 2023. Artificial Intelligence Based Chatbots to Combat COVID-19 Pandemic: A Scoping Review. (2023). https://doi.org/10.21203/rs.3.rs-2565141/v1

[9] Robert R Morris, Kareem Kouddous, Rohan Kshirsagar, and Stephen M Schueller. 2018. Towards an artificially empathic conversational agent for mental health applications: system design and user perceptions. *Journal of Medical Internet Research* 20 (2018), e10148. https://doi.org/10.2196/10148

[10] OpenAI. [n. d.]. OpenAI GPT API Documentation. https://beta.openai.com/docs/. Acesso em: 14/03/2024.

Received 19 Novembro 2024