

# UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

**ChatGPT na geração de artefatos de software: Um experimento envolvendo a aplicação de IA Generativa e as técnicas de engenharia de prompt no contexto do desenvolvimento de firmware.**

**Fernando Pereira Martines**

Trabalho de Conclusão de Curso  
MBA em Inteligência Artificial e Big Data

# **UNIVERSIDADE DE SÃO PAULO**

## **Instituto de Ciências Matemáticas e de Computação**

---

ChatGPT na geração de artefatos de  
software: Um experimento  
envolvendo a aplicação de IA  
Generativa e as técnicas de  
engenharia de prompt no contexto  
do desenvolvimento de firmware.

---

*Fernando Pereira Martines*

---

USP – São Carlos  
2024

Fernando Pereira Martines

# ChatGPT para a geração de artefatos de software: Um experimento envolvendo a aplicação de IA Generativa e as técnicas de engenharia de prompt no contexto do desenvolvimento de firmware.

Trabalho de conclusão de curso apresentado ao Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, como parte dos requisitos para obtenção do título de Especialista em Inteligência Artificial e Big Data.

Área de concentração: Inteligência Artificial.

Orientador: Prof. Dr. Ricardo Cerri.

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi  
e Seção Técnica de Informática, ICMC/USP,  
com os dados inseridos pelo(a) autor(a)

M385c      Martines, Fernando Pereira  
                ChatGPT para a geração de artefatos de software:  
                Um experimento envolvendo a aplicação de IA  
                Generativa e as técnicas de engenharia de prompt no  
                contexto do desenvolvimento de firmware. / Fernando  
                Pereira Martines; orientador Ricardo Cerri. -- São  
                Carlos, 2024.  
                76 p.

Trabalho de conclusão de curso (MBA em  
Inteligência Artificial e Big Data) -- Instituto de  
Ciências Matemáticas e de Computação, Universidade  
de São Paulo, 2024.

1. IA generativa. 2. ChatGPT. 3. Desenvolvimento  
de Firmware. 4. Geração de código. I. Cerri, Ricardo,  
orient. II. Título.

## DEDICATÓRIA

*À minha eterna esposa Sandra por  
todo amor, carinho, compreensão e  
apoio incansável nessa jornada.*

## AGRADECIMENTOS

Ao Instituto de Pesquisas Eldorado, pelo apoio e constante incentivo aos colaboradores pela busca por novos conhecimentos e pelo inspirador ambiente de inovação tecnológica.

À Profa. Dra. Solange Rezende, pelas incontáveis palavras de motivação em suas aulas e palestras, o que sempre nos trouxe confiança e vontade de aprender mais e mais.

Ao Prof. Dr. Ricardo Cerri, por todo apoio e orientação no processo de pesquisa e elaboração deste trabalho.

## EPÍGRAFE

"Inteligência artificial não é a inimiga da  
humanidade; é uma ferramenta para ampliar  
as capacidades humanas."

John McCarthy (2007)

## RESUMO

**MARTINES, F. P. ChatGPT para a geração de artefatos de software: Um experimento envolvendo a aplicação de IA Generativa e as técnicas de engenharia de prompt no contexto do desenvolvimento de firmware.** 2024. 74 f. Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

O presente trabalho explora o uso de IA generativa, com foco na ferramenta ChatGPT, no processo de geração de artefatos de software, tais como código-fonte, diagramas e casos de teste. A motivação principal é avaliar se essa tecnologia pode ser útil no desenvolvimento de firmware, um tipo de software fortemente acoplado ao hardware. O trabalho considera o potencial da IA generativa para automatizar tarefas repetitivas, aumentando a eficiência dos desenvolvedores, especialmente na criação de lógicas bem definidas e de uso recorrente. O experimento principal envolveu a geração de código em linguagem C para controlar um dispositivo eletrônico de monitoramento de temperatura. Para conduzir o experimento, foram utilizados conceitos da Engenharia de Prompt, que guiam a interação do usuário com o ChatGPT para otimizar a assertividade e a qualidade dos resultados gerados. O experimento foi estruturado em várias etapas, começando pela criação de estórias de usuário e especificações técnicas para orientar a IA na geração do código-fonte e de outros artefatos relacionados. Essas instruções foram enviadas ao ChatGPT através de prompts detalhados. Ao longo do processo, o código gerado foi analisado e diversas interações adicionais foram realizadas para corrigir pequenos erros apresentados. O código final foi considerado funcional e compatível com as especificações fornecidas, destacando a utilidade prática da ferramenta de IA. Os resultados demonstraram que, embora o ChatGPT seja eficaz na geração de código e outros artefatos, a intervenção humana continua sendo necessária para ajustes finos e refinamentos. Assim, a ferramenta não substitui completamente o papel do desenvolvedor, mas serve como um auxílio que pode aumentar a produtividade ao automatizar certas partes do processo de desenvolvimento. Conclui-se que ferramentas de IA generativa, como o ChatGPT, têm grande potencial para serem incorporadas ao desenvolvimento de firmware, acelerando a prototipação e o desenvolvimento da solução final através do apoio na geração automática dos principais artefatos de software envolvidos no processo.

Palavras-chave: IA generativa; ChatGPT; Desenvolvimento de Firmware; Geração de código.

## ABSTRACT

MARTINES, F. P. **ChatGPT for the generation of software artifacts: An experiment involving the application of Generative AI and prompt engineering techniques in the context of firmware development.** 2024. 74 f. Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

The present work explores the use of generative AI, with a focus on the ChatGPT tool, in the process of generating software artifacts, such as source code, diagrams, and test cases. The main motivation is to assess whether this technology can be useful in firmware development, a type of software that is closely integrated with hardware. The work considers the potential of generative AI to automate repetitive tasks, increasing developers' efficiency, especially in creating well-defined and frequently used logic. The main experiment involved generating code in C language to control an electronic temperature monitoring device. To conduct the experiment, concepts from Prompt Engineering were used, guiding user interaction with ChatGPT to optimize the accuracy and quality of the generated results. The experiment was structured in several stages, starting with the creation of user stories and technical specifications to guide the AI in generating source code and related artifacts. These instructions were sent to ChatGPT through detailed prompts. Throughout the process, the generated code was analyzed, and various additional interactions were made to correct small errors that appeared. The final code was deemed functional and compliant with the provided specifications, highlighting the practical usefulness of the AI tool. The results demonstrated that, although ChatGPT is effective in generating code and other artifacts, human intervention remains necessary for fine-tuning and refinements. Thus, the tool does not fully replace the role of the developer but serves as an aid that can increase productivity by automating certain parts of the development process. It is concluded that generative AI tools, such as ChatGPT, have great potential to be incorporated into firmware development, accelerating prototyping and the final solution's development by assisting in the automatic generation of key software artifacts involved in the process.

Keywords: Generative AI; ChatGPT; Firmware development; Code generation.

## LISTA DE ILUSTRAÇÕES

Figura 1 – Relação da IA e Aprendizado de Máquina. ....	12
Figura 2 – Firmware entre Hardware e Software. ....	17
Figura 3 – Placa de Desenvolvimento (devkit) utilizada nos experimentos.....	28
Figura 4 – Tela do Ambiente Integrado de Programação STM32CubeIDE. ....	28
Figura 5 – Tela principal do ChatGPT 4o. ....	29
Figura 6 – As etapas e o fluxo básico para realização dos experimentos.....	30
Figura 7 – Diagrama de blocos simplificado da solução pretendida.....	31
Figura 8 – Diagrama de classes da UML gerado pelo ChatGPT.....	44
Figura 9 – Diagrama de sequência da UML gerado pelo ChatGPT .....	46

## LISTA DE TABELAS

Tabela 1 – Comparativo das versões do ChatGPT .....	22
Tabela 2 – Estórias de usuário utilizadas no experimento. ....	32
Tabela 3 – Especificações gerais para operação na placa de desenvolvimento. ....	32
Tabela 4 – Demonstrações fornecidas ao ChatGPT com base em prompts <i>few-shot</i> .....	33
Tabela 5 – Preparação da interação com o ChatGPT com base dos artefatos da concepção. ..	34
Tabela 6 – Interação com ChatGPT para a entrada das estórias de usuário.....	34
Tabela 7 – Interação com ChatGPT para a entrada das especificações gerais.....	34
Tabela 8 – Interação com ChatGPT para a entrada das orientações e exemplos ( <i>few-shot</i> ). ...	35
Tabela 9 – Solicitação e resposta para geração de código-fonte em linguagem C.....	36
Tabela 10 – Descritivo da geração de código-fonte conforme a solicitação.....	36
Tabela 11 – 1 <sup>a</sup> solicitação para resolução de defeito em código-fonte gerado pelo ChatGPT.	37
Tabela 12 – 2 <sup>a</sup> solicitação para resolução de defeito em código-fonte gerado pelo ChatGPT.	38
Tabela 13 – Solicitação para melhoria no código-fonte gerado pelo ChatGPT. ....	40
Tabela 14 – Atualização das especificações gerais para melhoria da geração de código. ....	41
Tabela 15 – Solicitação para geração de código-fonte com a especificação atualizada.....	41
Tabela 16 – Descritivo da geração de código-fonte atualizada conforme a solicitação.....	42
Tabela 17 – Solicitação e resposta para geração de diagrama de classes feita ao ChatGPT....	43
Tabela 18 – Solicitação e resposta para geração de diagrama de sequência feita ao ChatGPT	45
Tabela 19 – Solicitação e resposta para geração de casos de teste feita ao ChatGPT.....	47

## LISTA DE ABREVIATURAS E SIGLAS

ADC	Analog-Digital Conversion/Converter
AIGC	Artificial Intelligence Generated Content
BDD	Behavior Driven Development
BERT	Bidirectional Encoder Representations from Transformers
CoT	Chain of Thoughts
DER	Diagrama Entidade-Relacionamento
DFD	Diagrama de Fluxo de Dados
FW	Firmware
GPT	Generative Pre-trained
GPU	Graphical Processing Unit
GCC	GNU Compiler Collection
GDB	GNU Debugger
GNU	GNU is Not Unix
HW	Hardware
IA	Inteligência Artificial
IAG	Inteligência Artificial Generativa
IDE	Integrated Development Environment
HAL	Hardware Abstraction Layer
HCI	Human-Computer Interface
LED	Light-Emitting Diode
LLM	Large Language Model
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
PCB	Printed Circuit Board
PLN	Processamento de Linguagem Natural
RNN	Recurrent Neural Networks
ROM	Read Only Memory
SysML	System Modeling Language
SW	Software
TTCN-3	Testing and Test Control Notation
UML	Unified Modeling Language
USE	UML-based Specification Environment

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>12</b>
1.1 Contextualização .....	12
1.2 Objetivos.....	14
1.3 Justificativa.....	15
1.4 Organização do texto .....	15
<b>2 REVISÃO BIBLIOGRÁFICA .....</b>	<b>16</b>
2.1 Conceitos Fundamentais.....	16
2.1.1 Firmware.....	16
2.1.2 PNL e os Modelos de Linguagem de Grande Escala .....	17
2.1.3 <i>Transformers</i> e o Surgimento dos Modelos de Atenção .....	18
2.1.4 Engenharia de <i>Prompt</i> e suas técnicas.....	20
2.2 Trabalhos relacionados .....	22
2.2.1 ChatGPT na tarefa de geração de artefatos de software.....	23
<b>3 METODOLOGIA.....</b>	<b>27</b>
3.1 Escopo e Definições Gerais .....	27
3.2 Etapas do Desenvolvimento .....	29
3.3 Projeto do Dispositivo Eletrônico .....	31
3.4 Estórias de Usuário e Especificações Gerais .....	32
<b>4 RESULTADOS .....</b>	<b>34</b>
4.1 Resultados da Geração de Código-fonte .....	35
4.2 Resultados da Geração de Diagramas .....	43
4.3 Resultados da Geração de Casos de Teste.....	47
<b>5 CONCLUSÃO.....</b>	<b>50</b>
<b>REFERÊNCIAS .....</b>	<b>52</b>
<b>ANEXO A – Interação inicial com o ChatGPT .....</b>	<b>55</b>
<b>ANEXO B – Código-fonte gerado inicialmente pelo ChatGPT .....</b>	<b>57</b>
<b>ANEXO C – Código-fonte aprimorado pelo ChatGPT .....</b>	<b>62</b>
<b>ANEXO D – Código-fonte corrigido pelo ChatGPT .....</b>	<b>63</b>
<b>ANEXO E – Código-fonte refeito pelo ChatGPT .....</b>	<b>66</b>
<b>ANEXO F – Código-fonte gerado pelo ChatGPT (especificação atualizada).....</b>	<b>70</b>

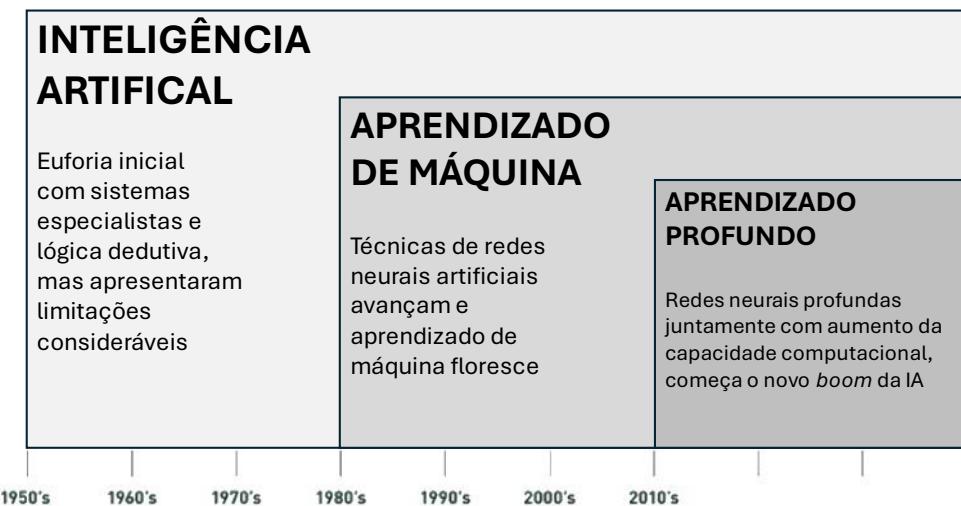
# 1 INTRODUÇÃO

## 1.1 Contextualização

Nos últimos anos tem se presenciado uma verdadeira explosão de soluções tecnológicas baseadas na aplicação de Inteligência Artificial (IA) nas mais diversas áreas. A aplicação da IA no contexto das inovações tecnológicas não é atual e tais esforços remontam meados do século XX, momento em que as pesquisas sobre o tema produziram modelos matemáticos e computacionais cuja aplicação produziram resultados práticos satisfatórios para a época.

O salto tecnológico das soluções atuais em relação às antigas se dá pelo fato da evolução das técnicas da IA, mais especificamente no âmbito do aprendizado de máquina e suas áreas mais especializadas, dentre elas, as técnicas de aprendizado profundo a partir da segunda década do século XXI. A Figura 1 mostra a relação os domínios desses conceitos indicando a especialização e localidade dentro dos temas da área de conhecimento da IA.

Figura 1 – Relação da IA e Aprendizado de Máquina.



Fonte: Adaptado de COPELAND (2016).

Associada à evolução das técnicas de aprendizado profundo, nota-se também a evolução das plataformas computacionais especializadas para determinados tipos de aplicação. Mais especificamente, as unidades de processamento gráfico (GPU's) tiveram um crescimento exponencial em performance e otimização, permitindo que modelos computacionais e algoritmos mais complexos operando sobre um maior volume de dados pudessem computados em tempo aceitável para a viabilização prática das soluções comerciais envolvendo IA.

Com o avanço dos estudos dos modelos de linguagens de grande escala (LLM's) – uma área de pesquisas e tecnologias do processamento de linguagem natural (NLP) – e a sua viabilização comercial a partir do uso dessas plataformas computacionais modernas, como GPU's e serviços em nuvem, a sociedade atual se vê imersa em uma nova onda de inovações e soluções apoiadas por IA, que agora se destaca pela sua notável capacidade de geração de conteúdo multimídia a partir de interações conversacionais feitas pelo usuário.

Essa nova categoria de técnicas de IA – denominada IA Generativa (IAG) – tem provocado na sociedade como um todo, vislumbres, ceticismos, discussões e até regulamentações por ter atingido considerável destaque a partir do seu inegável potencial de revolucionar as aplicações tecnológicas em qualquer área que se queira imaginar. A capacidade de geração de conteúdo das aplicações baseadas em IA Generativa pode potencializar e aumentar a capacidade dos seres humanos nas mais diversas atividades.

Dentro do contexto da geração de textos, especialidade primária das IA's Generativas, há a possibilidade explorá-las para a criação de conteúdos próprios para as mais diversas categorias textuais, tais como produção de resumos, roteirização, textos analíticos, geração de *insights*, análises explicativas, obras literárias etc. enfim, uma infinidade de possibilidades.

A interface humano-computador (HCI) mais comum para uso das IA's Generativas em computadores por usuários em geral são baseadas em *prompts*, que são interfaces baseadas em entrada de texto digitado pelo usuário da aplicação visando interagir com o sistema para seus propósitos. A partir dessa interação do usuário com o computador, ocorre o processamento dos algoritmos sobre os dados textuais codificados e a produção dos resultados normalmente na forma de conteúdo multimídia baseados na solicitação do usuário.

Dado o conhecimento que se tem sobre o funcionamento interno dos LLM's, a interação do usuário com tais sistemas pode ser melhorada no intuito de tornarem mais assertivos os resultados obtidos a partir do *prompt* informado a esses sistemas. Este uso mais sistemático do mecanismo de *prompts* convencionou-se chamar de *Engenharia de Prompt*, que oferece técnicas para a elaboração e estruturação de textos de entrada a fim de, digamos, instruir e induzir a operação dos LLM's a direcionarem melhor os resultados conforme o esperado.

Uma categoria particular de textos produzidos para propósitos específicos são as linguagens de programação de computadores, cujo conteúdo é fortemente regulado pela gramática e combinação de símbolos dispostos em uma sintaxe formal e bem definida para manipulação por máquinas. Tais linguagens de programação são utilizadas por desenvolvedores de software para criar programas com instruções (código-fonte) que serão executadas por

computadores após um processo de transformação desse código-fonte em um programa descrito em linguagem de máquina.

O uso da IA Generativa com o objetivo de apoiar a geração de código-fonte tem sido explorado recentemente devido às características propícias desse tipo de conteúdo textual. As linguagens de programação, por serem compostas por palavras-chaves bem definidas, podem também vir a ser de conhecimento dos modelos de linguagem de grande escala.

Este trabalho visa explorar, através de um experimento com *prompts* direcionados à uma aplicação IA Generativa bastante conhecida – o ChatGPT –, quais são os resultados da geração de código-fonte em linguagem de programação C para a implementação de um firmware, um tipo especializado de software bastante acoplado ao sistema de hardware.

## 1.2 Objetivos

### 1.2.1 Objetivo Geral

O objetivo geral deste trabalho é realizar um experimento utilizando as técnicas de engenharia de *prompt* que permita se ter uma avaliação sobre a utilidade e potenciais aplicações das ferramentas de IA Generativa para o apoio na geração de código-fonte e outros artefatos de software auxiliares no contexto do desenvolvimento de firmware para dispositivos eletrônicos.

### 1.2.2 Objetivos específicos

Dentre os objetivos específicos desse trabalho estão:

- Determinar o problema-alvo a ser explorado no experimento;
- Conseguir descrever os requisitos do problema-alvo na forma de estórias de usuário;
- Criar *prompts* a partir de técnicas para guiar o processo de geração de artefatos;
- Conseguir obter uma proposta de solução factível a partir dos *prompts* processados;
- Gerar os artefatos da solução proposta por meio de uma ferramenta de IA Generativa;
- Analisar a assertividade das respostas refinando o contexto e solicitações fornecidos;
- Avaliar os resultados com base na aplicabilidade e completude dos artefatos gerados;
- Levantar os eventuais problemas e limitações observadas durante o experimento;
- Levantar os aspectos mais importantes a serem considerados para a realização de novos experimentos com base na atualização dos modelos e outras ferramentas.

### 1.3 Justificativa

A principal motivação para a realização desse trabalho é a de se verificar a possibilidade das ferramentas de IA Generativa passarem a ser empregadas no processo de desenvolvimento de software, mais especificamente no desenvolvimento de firmware, como um mecanismo automático para geração de código-fonte de lógicas comuns, ou seja, aquelas as quais já se tem um conhecimento estabelecido mas cuja implementação em microcontroladores ainda varia devido à variedade de fornecedores de chip e ao forte acoplamento deste tipo de software ao hardware subjacente ao qual irá operar. Adicionalmente, serão verificados os resultados da geração de outros artefatos, tais como diagramas para documentação da especificação e *design* do firmware e casos de teste para apoio às atividades de verificação e validação de software.

### 1.4 Organização do texto

No capítulo 2 é apresentada a fundamentação teórica sobre Firmware, IA Generativa e engenharia de *prompt*, apresentando os conceitos fundamentais, a revisão das principais técnicas, modelos e trabalhos atuais e mais relevantes na literatura.

No capítulo 3 é descrita a metodologia a ser aplicada, detalhando os passos a serem considerados para o desenvolvimento do experimento e a discussão sobre quais das principais técnicas de engenharia de *prompt* podem ser aplicadas dado o contexto do problema.

No capítulo 4 são apresentados os resultados obtidos na interação com uma ferramenta da IA Generativa. Tais resultados são decorrentes do processamento dos *prompts* elaborados e são avaliados conforme a sua aplicabilidade imediata em projetos, ou seja, o grau de prontidão dos resultados serem diretamente empregados da construção da solução ou se eventualmente são necessárias modificações para que se obtenha a solução completa.

Por fim, no capítulo 5 são apresentadas as considerações finais e a conclusão sobre os experimentos trazendo uma percepção sobre a efetividade de aplicar tais ferramentas como apoio à geração de artefatos de *software* em relação à uma abordagem manual de construção feita por desenvolvedores especializados, destacando seus potenciais e suas limitações, além de possíveis indicações de trabalhos futuros.

## 2 REVISÃO BIBLIOGRÁFICA

### 2.1 Conceitos Fundamentais

#### 2.1.1 Firmware

Firmware é uma combinação de hardware e software que reside em um chip ou em um conjunto de chips dentro de um dispositivo (PRESSMAN, 2011). Ele é responsável por controlar funções essenciais e específicas do dispositivo, atuando como o software operacional que interage diretamente com o hardware.

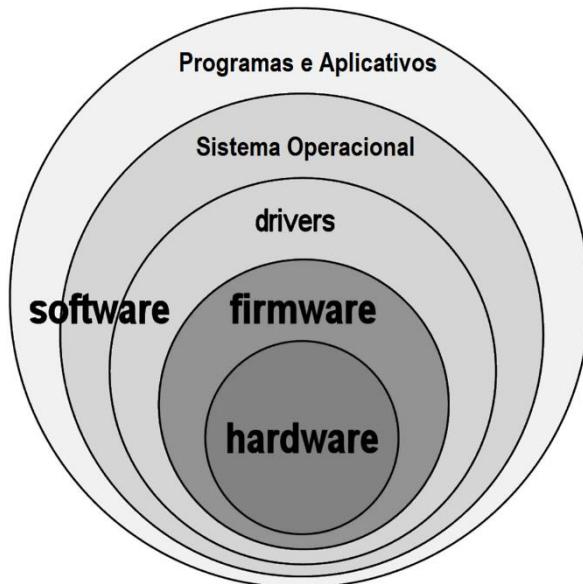
Na literatura, há alguns autores que deixam explícita a relação simbiótica do firmware com o hardware classificando-o como uma categoria distinta de software. Outros consideram o firmware como um tipo especializado de software.

Tanenbaum (2009) descreve como sendo uma ponte entre o hardware e o software de alto nível, facilitando a comunicação e operação eficiente do dispositivo. Stallings (2013) afirma que firmware é o software especializado embutido em dispositivos de hardware para controlar suas funções básicas e permitir sua operação correta. Morimoto (2017) diz que firmware é aquele geralmente armazenado em memória não volátil, como ROM ou *flash*, para garantir que o software permaneça no dispositivo mesmo quando ele é desligado. A Figura 2 mostra a localização do firmware estando diretamente acoplado ao hardware e fazendo a interface entre este e o software de alto nível.

Pode-se afirmar que firmware é, portanto, um tipo de software especializado que está embutido em dispositivos de hardware para controlar suas funções básicas e operá-los corretamente. Ele é projetado e construído para interagir diretamente com o hardware, servindo como uma ponte entre o software de mais alto nível (como o sistema operacional) e os componentes físicos do dispositivo.

Por essas características, a geração de código-fonte para desenvolvimento do firmware deve levar em conta não apenas o conhecimento sobre a linguagem de programação utilizada e a estrutura e organização de computadores de propósito geral, mas também sobre os detalhes (definições e características do circuito eletrônico, temporizações e interfaces de periféricos e outros chips do circuito eletrônico) do hardware ao qual o firmware irá operar.

Figura 2 – Firmware entre Hardware e Software.



Fonte: Adaptado de HEKLER (2020).

### 2.1.2 PNL e os Modelos de Linguagem de Grande Escala

A evolução dos modelos de linguagem natural culminou na criação dos modelos de linguagem de grande escala que revolucionaram diversas aplicações computacionais. Esses modelos são baseados em arquiteturas de aprendizado profundo, como redes neurais, e treinados com enormes quantidades de dados textuais.

O PNL começou a ganhar relevância na década de 1950 com a pesquisa em tradução automática e a ideia de usar computadores para compreender e comunicar em linguagem humana. A criação do "*Test Turing*" na década de 50 (TURING, 1950) foi um marco que destacou a possibilidade de uma máquina exibir comportamento considerado inteligente, capaz de se comunicar em linguagem natural tão bem que a ponto de não se tornar distinguível de outro ser humano fluente na língua.

Nos anos seguintes, pesquisadores desenvolveram métodos baseados em regras e gramáticas formais, como os sistemas baseados em regras de produção, que eram essencialmente conjuntos de regras predefinidas para manipular *strings* de texto, tais como o sistema DENDRAL (FEIGENBAUM; BUCHANAN; LEDERBERG, 1971) e o MYCIN (SHORTLIFFE, 1976). No entanto, esses sistemas tinham limitações severas ao lidar com a variabilidade e ambiguidade da linguagem humana.

A solução veio com o avanço do aprendizado de máquina na década de 1990 e 2000, com o uso mais amplo de modelos estatísticos e o advento das redes neurais artificiais.

Nos anos 90, a transição para abordagens estatísticas marcou um ponto de virada. Os modelos baseados em aprendizado supervisionado começaram a ser usados para tarefas de classificação de texto e análise de sentimentos. Técnicas de *Machine Learning* (ML) permitiram que os modelos fossem ajustados com base em grandes volumes de dados anotados, o que aumentou significativamente a precisão dos sistemas de PNL (JURAFSKY; MARTIN, 2009).

Em paralelo, o desenvolvimento das redes neurais e o advento do aprendizado profundo – *Deep Learning* – a partir da década de 2010 elevara o PNL a um novo patamar. Arquiteturas como as redes neurais recorrentes (RNN's) e, posteriormente, as redes de memória de longo prazo (LSTM) foram usadas para lidar com a sequência e a dependência contextual nas tarefas de linguagem (GRAVES, 2012).

Os LLM's continuaram a evoluir rapidamente e modelos como o GPT-4 e outros competidores, como o PaLM da Google e o LLaMA da Meta, aumentaram ainda mais o número de parâmetros, melhoraram a eficiência do treinamento e reduziram o viés. A exploração de modelos multimodais, que integram dados de texto, imagem e áudio, é um campo emergente que promete ampliar ainda mais as capacidades desses sistemas para uso de propósito geral.

No entanto, o avanço dos LLM's levantou também desafios éticos e práticos, como o consumo elevado de recursos computacionais e as implicações do viés nos dados de treinamento. O futuro dos LLM's provavelmente envolverá uma maior regulamentação e a busca por modelos mais eficientes e sustentáveis (BENDER et al., 2021).

### 2.1.3 *Transformers* e o Surgimento dos Modelos de Atenção

A verdadeira revolução do uso dos LLM's veio com a introdução do mecanismo de atenção e, mais significativamente, com o modelo *Transformer*, apresentado por Vaswani et al. em 2017. O Transformer permitiu que o processamento da linguagem natural fosse escalado de maneira sem precedentes devido à sua capacidade de parallelizar o processamento e capturar relações contextuais em grandes sequências de texto (VASWANI et al., 2017).

Ao contrário das RNN's e LSTM's, que processam a informação sequencialmente, o Transformer utiliza um mecanismo de auto-atenção para considerar todas as palavras de uma frase simultaneamente, permitindo que o modelo construa uma compreensão mais rica e profunda do contexto. Essa inovação possibilitou posteriormente a criação de modelos muito

maiores e mais poderosos, como o BERT (*Bidirectional Encoder Representations from Transformers*) e o GPT (*Generative Pretrained Transformer*).

O BERT, desenvolvido pelo Google em 2018, trouxe um novo paradigma para o NLP, introduzindo o conceito de pré-treinamento bidirecional, em que o modelo aprende representações contextuais a partir de todas as palavras de uma sequência em vez de processar o texto de forma unidirecional. Isso permitiu que o BERT alcançasse um desempenho sem precedentes em várias tarefas de NLP, como classificação de textos e resposta a perguntas (DEVLIN et al., 2018).

Simultaneamente, o GPT, desenvolvido pela OpenAI, se destacou pelo uso de grandes quantidades de dados e pela sua abordagem de pré-treinamento não supervisionado. A segunda versão, GPT-2, e a subsequente GPT-3, lançada em 2020, foram treinadas em bases de dados massivas, contendo centenas de bilhões de parâmetros, e mostraram uma capacidade impressionante de gerar texto coeso e realista. O GPT-3, em particular, alcançou notoriedade por sua habilidade de realizar uma ampla gama de tarefas linguísticas sem a necessidade de treinamento adicional (BROWN et al., 2020).

Na base da operação de ambos os modelos estão os Transformers. Eles, portanto, revolucionaram o processamento de linguagem natural ao introduzir essa nova abordagem para modelar sequências de texto utilizando um mecanismo de atenção que permite ao modelo considerar todas as palavras de uma sequência ao mesmo tempo, capturando melhor as relações de longo alcance no texto, em vez de processar as entradas de maneira sequencial.

A base do Transformer é o mecanismo de auto-atenção, que permite que o modelo dê "atenção" a diferentes partes de uma sequência de entrada ao calcular suas representações. O Transformer é composto por duas partes principais: o codificador (*encoder*) e o decodificador (*decoder*), embora em muitos casos de NLP (como no BERT), apenas o codificador seja utilizado.

O codificador do Transformer é responsável por transformar a sequência de entrada em uma série de representações internas (conhecidos como vetores de *embeddings*). Ele é composto por múltiplas camadas idênticas que consistem em duas subcamadas principais:

1. Mecanismo de Auto-atenção: O componente-chave que permite ao modelo considerar todas as palavras da sequência de entrada de maneira simultânea. Esse mecanismo calcula uma pontuação de importância entre cada par de palavras na sequência, identificando quais palavras são mais relevantes umas para as outras.

**2. Feed-Forward Neural Network:** Após passar pelo mecanismo de auto-atenção, a saída passa por uma rede neural simples para transformar as representações contextuais em vetores de dimensão fixa.

Cada uma dessas subcamadas é acompanhada por uma conexão residual, que ajuda a mitigar problemas de gradientes durante o treinamento.

O decodificador, semelhante ao codificador, é composto de três camadas que incluem:

1. Mecanismo de auto-atenção: Focado na saída parcial gerada até o momento, determinando quais palavras pré-geradas são mais relevantes.

2. Atenção Cruzada (*Cross-Attention*): Este mecanismo observa as representações produzidas pelo codificador para integrar informações sobre a entrada na geração de saída.

3. *Feed-Forward Neural Network*: Assim como no codificador, esta etapa transforma as representações intermediárias em vetores de dimensão fixa.

O núcleo do Transformer é o mecanismo de atenção, em especial, a atenção escalada por produto interno (*Scaled Dot-Product Attention*). Em uma visão simplificada, cada palavra da sequência de entrada é associada a três vetores: *Query* (Q), que representa a palavra que está sendo considerada; *Key* (K), que captura as características das palavras que servem como possíveis "chaves" relevantes; e *Value* (V), que contém a informação real a ser propagada para as representações finais.

A atenção é calculada como o produto interno entre o vetor consulta (Q) de uma palavra e o vetor chave (K) de outra palavra, escalado por uma constante para evitar que os valores explodam. O resultado é passado por uma função *Softmax*, que normaliza os pesos de atenção. Finalmente, os pesos de atenção são usados para ponderar os valores (V) correspondentes e gerar a saída ponderada (VASWANI et al., 2017).

Os Transformers mudaram drasticamente o campo do PNL ao introduzir um método paralelo e altamente eficiente de processar texto, baseado em auto-atenção. Ao permitir que o modelo considere simultaneamente todas as palavras de uma sequência, ele consegue capturar melhor as nuances da língua e suas dependências de longo alcance durante o processamento. Essa abordagem se provou eficaz em uma ampla variedade de tarefas de PNL, desde tradução automática até a geração automática de texto.

#### 2.1.4 Engenharia de *Prompt* e suas técnicas

Engenharia de *Prompt* é o conjunto de técnicas utilizadas em LLM's na qual as entradas ou comandos fornecidos ao modelo são cuidadosamente formulados para direcionar suas

respostas de maneira mais eficiente e específica. Essa prática envolve a criação de instruções detalhadas e bem estruturadas, frequentemente utilizando exemplos e padrões, a fim de otimizar a precisão e a coerência das saídas geradas pelos modelos.

Segundo Liu et al. (2023), a Engenharia de *Prompt* é o processo de projetar e refinar comandos ou perguntas apresentados a um modelo de linguagem com o objetivo de obter respostas mais relevantes, precisas ou úteis, explorando as capacidades implícitas do modelo. Essa técnica tem se mostrado especialmente valiosa na adaptação de modelos de linguagem para tarefas específicas, permitindo que os LLM's executem funções complexas sem necessidade de retreinamento extensivo.

Existem diversas técnicas de engenharia de *prompt* sendo estudadas e relatadas na literatura atualmente. Dentre aquelas de maior destaque e de uso mais amplo, estão as seguintes:

1. *Zero-Shot Prompting* (zero disparos): Nesta técnica, os LLM's são guiados por *prompts* sem que sejam fornecidos exemplos específicos de *input-output*, utilizando-se do conhecimento pré-existente do modelo para completar tarefas sem treinamento adicional (WEI et al., 2021). Essa técnica permite que os modelos realizem novas tarefas sem a necessidade de grandes volumes de dados rotulados.

2. *Few-Shot Prompting* (poucos disparos): Ao contrário do *zero-shot*, a técnica *few-shot* fornece ao modelo alguns exemplos, o que o ajuda a entender a tarefa. Essa abordagem tem se mostrado eficaz em tarefas mais complexas, melhorando o desempenho dos modelos de linguagem com base em poucos exemplos de alta qualidade (BROWN et al., 2020).

3. *Chain-of-Thought (CoT) Prompting*: CoT é uma técnica que orienta os LLM's a seguir cadeias lógicas de raciocínio, estruturando melhor o processo de tomada de decisão. WEI et al. (2022) demonstraram que essa abordagem é eficaz em problemas de raciocínio complexo, melhorando o desempenho dos modelos em *benchmarks* de raciocínio lógico e matemático.

4. *Auto-CoT Prompting*: Desenvolvida por Zhang et al. (2022), essa técnica automatiza a criação de cadeias de raciocínio, substituindo a construção manual por *prompts* como "vamos pensar passo a passo". Através da amostragem diversificada, o Auto-CoT melhora a robustez e reduz os erros, melhorando o aprendizado com poucos exemplos.

5. *Self-Consistency*: Wang et al. (2022) introduziram a auto-consistência como uma estratégia de decodificação que aumenta o desempenho do raciocínio em CoT. Essa técnica gera múltiplas cadeias de raciocínio diversificadas e escolhe a resposta final mais consistente, resultando em melhorias significativas de precisão em vários *benchmarks*.

6. *MathPrompter*: Desenvolvida para tarefas de resolução de problemas matemáticos, essa técnica estrutura a solução de problemas em etapas verificáveis, aumentando a precisão

dos resultados, especialmente em problemas que exigem validação de passos intermediários (IMANI et al., 2023).

Essas técnicas, ao guiarem os LLM's em tarefas complexas, mostram como a engenharia de *prompts* pode moldar e melhorar a interação com modelos de linguagem de maneira adaptada ao contexto da tarefa. Para a realização do experimento desse trabalho, em linha com as características do uso de LLM's para geração de texto especializado (linguagem de programação ou linguagens específicas de domínio), será explorada a técnica de *few-shot prompting*, considerada mais adequada para entradas de texto e contexto complexo representado na forma de estórias de usuário e requisitos.

## 2.2 Trabalhos relacionados

Devido ao fato do lançamento das principais ferramentas de IA Generativa terem ocorrido há aproximadamente dois ou três anos, os trabalhos relacionados à aplicação dessas ferramentas no contexto da geração de código-fonte são bastante recentes.

Esse trabalho é baseado exclusivamente no ChatGPT, a solução da IA Generativa da OpenAI, lançado como solução comercial ao público geral em 2022. Portanto, os trabalhos oriundos da revisão bibliográfica levam em conta apenas essa ferramenta.

Wu et al (2023) trouxeram em seu artigo os dados comparativos das diversas versões disponíveis do GPT, como pode ser visto na Tabela 1. A OpenAI transformou em solução comercial as versões 3 e 4 no formato de *chat*.

Tabela 1 – Comparativo das versões do ChatGPT

	GPT-1	GPT-2	GPT-3	GPT-4
Data da liberação	Junho/2018	Fevereiro/2019	Maio/2020	Março/2023
Parâmetros do modelo	117 milhões, 12 camadas, 768 dimensões	1,5 bilhão, 48 camadas, 1.600 dimensões	175 bilhões, 96 camadas, 12.888 dimensões	Não publicado
Janela de contexto	512 tokens	1024 tokens	2048 tokens	8195 tokens
Tamanho dos dados pré-treinados	~ 5 GB	~ 40 GB	~ 45 TB	Não publicado
Origem dos dados	BooksCorpus, Wikipedia	WebText	Common Crawl etc.	Não publicado
Técnica de aprendizado	Não Supervisionado	Aprendizado Multitarefa	Aprendizado em Contexto	Aprendizado Multimodal

Fonte: Wu et al. (2023)

Os dados mostram que o crescimento da capacidade do modelo GPT entre as versões teve um aumento exponencial, o que o tornou atualmente uma das maiores aplicações de IA Generativa em uso.

Wu et al. (2023) também listaram pontos positivos e negativos do ChatGPT baseados em experiências de uso e em informações públicas divulgadas pela OpenAI. Como pontos fortes e vantagens, o ChatGPT apresenta: (1) capacidade de geração de textos e entendimento multimodal, (2) forte habilidade de raciocínio e criatividade rica, e (3) planejamento e modelagem do conhecimento. Como pontos fracos e desvantagens: (1) erros factuais e alucinações, (2) modelagem insuficiente do conhecimento explícito, e (3) os custos da pesquisa e desenvolvimento são altos (WU et al., 2023). Embora a percepção a partir do uso do ChatGPT possa trazer a ideia de um sistema, de fato, inteligente, ele ainda não deixa de ser um tipo de ferramenta de IA para geração de conteúdo (AIGC).

### 2.2.1 ChatGPT na tarefa de geração de artefatos de software

Nesses últimos anos, após o lançamento das soluções de IA Generativa por grandes corporações e acessíveis ao público em geral por meio de serviços em nuvem, vários trabalhos surgiram relatando as análises provenientes das suas experiências na tarefa de usar a IA Generativa e os LLM's especificamente para a geração de artefatos de software de alto nível, o tipo de software que é escrito usualmente em linguagens de programação de alto nível e são executadas ou em máquina virtual ou em computadores contendo processadores de propósito geral, como, por exemplo, as linguagens Java e Python, majoritariamente.

Previamente ao lançamento oficial do ChatGPT pela OpenAI como uma ferramenta de software acessível ao público, a iniciativa mais relevante de treinar LLM's e aplicar IA Generativa para a finalidade de geração de código foi o CodeBERT (FENG et al., 2020), um modelo pré-treinado e construído com base em conjuntos de dados de busca por código, tais como CodeSearchNet (HUSAIN et al., 2019), que visou aprender representações textuais de propósito geral baseado em uma abordagem bimodal, oriundas de linguagens de programação e linguagem natural. O CodeBERT foi construído com base na arquitetura de rede neural profunda dos Transformers (VASWANI et al., 2017) e treinado com códigos de vários repositórios de código do Github. Ele suporta a geração de documentação a partir do código e transcrição de código em até seis linguagens de programação (FENG et al., 2020).

No contexto do uso do ChatGPT, Guo (2024) realizou um experimento que explorou o potencial de geração de código do ChatGPT na tentativa de se criar uma aplicação Web em

linguagem Java a partir de requisitos descritos em linguagem natural. Seus resultados demonstraram que o ChatGPT conseguiu produzir uma aplicação web para o login (acesso a um portal com base em credenciais de usuário e senha informados em uma página web) a partir da interação de *prompts* direcionados. Além da completa funcionalidade do código gerado, que foi posto em execução em um servidor de aplicações real, tal código foi avaliado como de alta legibilidade e qualidade, podendo ser aplicado, portanto, diretamente em projetos (GUO, 2024). Esse resultado reforça o fato da possibilidade de uso do ChatGPT, ou mais amplamente até das ferramentas de IA Generativa disponíveis, como assistentes efetivos dos desenvolvedores de software.

Diversos outros trabalhos abordaram quantitativamente as habilidades da IA Generativa, e mais especificamente do ChatGPT, na geração de código-fonte de programas de computador. A maioria deles buscaram analisar a consistência e a qualidade do código gerado. Clark et al. (2024), por exemplo, avaliaram 625 códigos em linguagem Python gerados a partir de *prompts* indicando que o ChatGPT é capaz de gerar bons códigos, embora contenham defeitos – como são também aqueles feitos por humanos – necessitando de revisões e correções manuais.

Os trabalhos de Yan et al. (2023) e Moratis et al. (2024) trouxeram análises sobre a consistência e qualidade do código gerado a partir de *prompts*. O segundo acrescentou ainda a análise sobre a aderência a padrões, estilo de codificação e melhorias práticas. Em ambos os trabalhos foram feitas ressalvas sobre a importância da qualidade do *prompt* dirigido ao ChatGPT, que pode ter impacto direto na efetividade do código gerado como resultado. Foi nessa linha da qualidade dos *prompts* que Lui et al. (2024) elaboraram um construtor de *prompts* que teve como objetivo melhorar a performance da geração de código e demonstraram, com uma série de experimentos, que a efetividade do código realmente aumenta com *prompts* de alta qualidade baseado em técnicas de engenharia de *prompt*.

Rahmaniar et al. (2024) confirmaram também o potencial do ChatGPT destacando que a relação simbiótica entre IA Generativa e desenvolvimento de software será mais robusta, complexa e transformativa. No entanto, trouxeram à tona a necessidade de reflexão na comunidade de software sobre implicações éticas do uso massivo dos modelos da IA Generativa, uma vez que consideram que o desenvolvimento de software por si só não trata apenas da geração de código em si – tarefa pela qual máquinas treinadas com volume gigantesco de dados são notoriamente capazes de fazer –, mas da cognição aplicada a solução de problemas reais por meio da concepção, construção e uso efetivo de sistemas computacionais.

Não apenas na geração de código-fonte as ferramentas de IA Generativa podem auxiliar no processo de desenvolvimento de software, elas podem também ser aplicadas na geração de outros artefatos auxiliares provenientes das etapas de análise, projeto e verificação de software.

No âmbito da análise e projeto, a IA Generativa pode ser usada para elaboração de diagramas visuais gerados a partir de notação textual.

A UML (OMG, 2017) é uma linguagem padrão utilizada para especificar, elaborar, visualizar e documentar artefatos de sistemas de software. Ela fornece uma especificação para se criar diagramas que representam a estrutura e o comportamento de um sistema de software, permitindo que desenvolvedores e engenheiros de software comuniquem ideias e conceitos de maneira clara e consistente. A UML é amplamente utilizada na indústria de software como padrão *de facto* para a criação de diagramas. Assim como as linguagens de programação, ela possui especificações formais e bem definidas que permitem a manipulação e interpretação por máquinas para a geração de diagramas a partir de texto.

Os trabalhos de Combemale et al. (2023) e Camara et al. (2023) exploraram as capacidades do ChatGPT no contexto da modelagem de software. Camara et al. (2023) fizeram experimentos para testar na prática a geração de diagramas da UML pelo ChatGPT. Vale destacar que a versão 3.5 do ChatGPT não possui a capacidade de geração de conteúdo imagético, tais como ilustrações, figuras ou diagramas. Isso é possível apenas na versão 4 (paga) da ferramenta. Portanto, a geração de diagramas pode ser feita em notação textual usando composição de caracteres simples ou em formato legível para outras ferramentas, tais como PlantUML, USE, Yuml, Markdown UML, Mermaid e UMLEt (Camara et al., 2023).

Essa capacidade de geração de diagramas UML a partir de notação textual permite que o ChatGPT seja também explorado para a elaboração desse tipo de artefato durante as etapas de projeto e elaboração da solução de software, em apoio aos arquitetos, projetistas e programadores de software. Esse trabalho explora a geração de diagramas para o problema proposto no intuito de verificar os resultados da sua aplicação para o *design* do firmware.

Por fim, referindo-se a etapa inicial de levantamento de requisitos de qualquer sistema ou produto – em que as necessidades e expectativas dos usuários são capturadas e registradas para darem origem à propostas de solução em artefatos de engenharia ou sistemas computacionais –, tem-se que a IA Generativa pode também apresentar a capacidade de interpretar texto contigo em estórias de usuário e casos de uso para determinadas finalidades, sejam essas de transformação para outras categorias de conteúdo textual, de análise de aspectos específicos ou mesmo de revisão e geração de *insights*.

Nesse contexto, Aamana et al. (2024) demonstraram em experimentos o potencial de avaliação da qualidade da escrita de estórias de usuário pelo ChatGPT com base nas técnicas de engenharia de *prompt*. Como estórias de usuário, no geral, são textos descritos em linguagem natural usualmente baseado em convenções bem estabelecidas acerca de capacidades e restrições desejadas para a solução, a técnica de engenharia de *prompt few-shot* mostrou-se a mais promissora na tarefa de avaliar a qual categoria de requisitos de qualidade determinadas estórias de usuário pertencem, indicam ou implicam (Aamana et al., 2024).

Os experimentos que serão descritos no Capítulo 5 desse trabalho também exploram estórias de usuário em entradas de texto para a IA Generativa. Porém, diferentemente de Aamana et al. (2024), ele não visa extrair requisitos e aspectos de qualidade das estórias de usuário, mas de usá-las como contexto para solicitações de geração de código-fonte que atendam a requisitos presentes nessas estórias, assim como a geração de outros artefatos no âmbito do desenvolvimento direcionado a firmware que, como dito, é uma categoria especializada de software fortemente acoplada ao hardware.

Portanto, esse trabalho explora o potencial de geração de conteúdo textual da IA Generativa, mais especificamente do ChatGPT, na transformação de estórias de usuários e requisitos dados como entrada em outros artefatos de software baseados em texto, tais como diagramas em notação textual, código-fonte de programas descritos em linguagem de programação no domínio do firmware e roteiros de testes para verificação de software usando a técnica de engenharia de *prompt few-shot* para avaliação de seus potenciais e limitações no apoio ao desenvolvimento de firmware.

## 3 METODOLOGIA

### 3.1 Escopo e Definições Gerais

O desenvolvimento dos experimentos desse trabalho baseou-se em um cenário hipotético de projeto de um dispositivo eletrônico considerando os artefatos de software minimamente previstos para as etapas de concepção, elaboração, construção e testes. Tal cenário é descrito em detalhes na seção 3.3.

A proposição em analisar o potencial de utilidade de ferramentas de IA Generativa se apoiou na ideia de que os analistas de requisitos, donos de produtos (*Product Owners*) ou mesmo desenvolvedores possam fornecer ao ChatGPT os textos elaborados relacionados à atividade de levantamento e especificação de requisitos da solução a ser desenvolvida.

Devido à natureza intrínseca de tais requisitos serem usualmente descritos em linguagem natural, viu-se a oportunidade imediata de aplicar ferramentas baseadas em LLM's na transformação desses textos de origem em textos produzidos em outras linguagens, tais como a linguagem de notação gráfica para diagramas, linguagem de programação para geração de código-fonte e novamente linguagem natural para descrição de casos de teste.

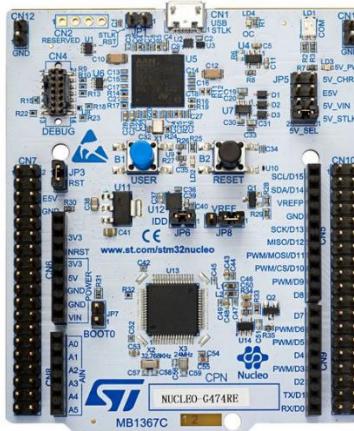
A versão do ChatGPT utilizada nos experimentos foi a versão 4o, a mais recente disponibilizada comercialmente pela OpenAI ao público. Pelo que pode ser observado através dos dados da Tabela 1, essa versão apresenta performance e capacidade bem superiores em relação às versões anteriores. O que se esperou obter, portanto, de partida e como consequência, foi o aumento do potencial e da assertividade dos resultados devido ao maior número de *tokens* processados e ao maior número de parâmetros disponíveis.

Para a avaliação dos resultados dos artefatos produzidos nos experimentos foi utilizada uma placa de desenvolvimento (*devkit*) disponível para aquisição através da internet. Tal placa de desenvolvimento foi projetada e disponibilizada ao público pelo próprio fabricante do chip microcontrolador, podendo esta ser utilizada para acelerar o processo de desenvolvimento e verificações prévias do firmware enquanto a placa de circuito impresso (PCB) do sistema eletrônico da solução final pretendida é projetada, fabricada e testada em paralelo, conforme previsto em um ciclo convencional de desenvolvimento de um produto baseado em sistemas microcontrolados.

A Figura 3 mostra a foto de uma das faces da placa de desenvolvimento cujo *Part Number* é o NUCLEO-G071RB, da fabricante ST Microelectronics. Ela possui um

microcontrolador baseado na microarquitetura da família ARM® Cortex-M™ com núcleo de processamento de propósito geral e diversos periféricos integrados em seu próprio chip.

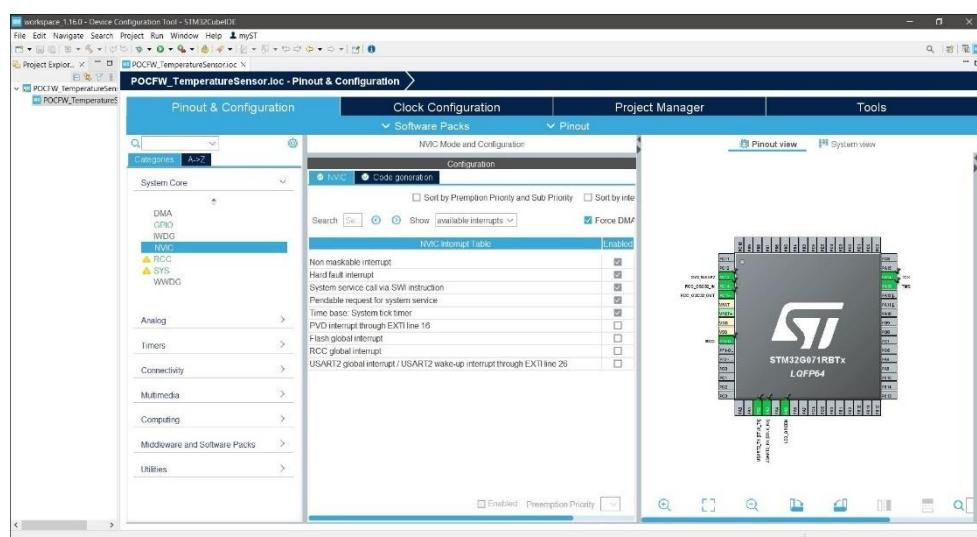
Figura 3 – Placa de Desenvolvimento (devkit) utilizada nos experimentos.



Fonte: Site da ST Microelectronics.<sup>1</sup>

Como ambiente de desenvolvimento integrado (IDE) para a compilação do código-fonte e execução do firmware na placa de desenvolvimento foi utilizado o *STM32CubeIDE*, que permite a criação de projetos de firmware por desenvolvedores integrando o compilador de código aberto GNU GCC e G++, para código em linguagem C e C++, respectivamente, além de suporte à depuração com o GNU GDB e várias outras atividades previstas na programação de microcontroladores deste fabricante. A Figura 4 mostra a tela principal desta IDE.

Figura 4 – Tela do Ambiente Integrado de Programação STM32CubeIDE.



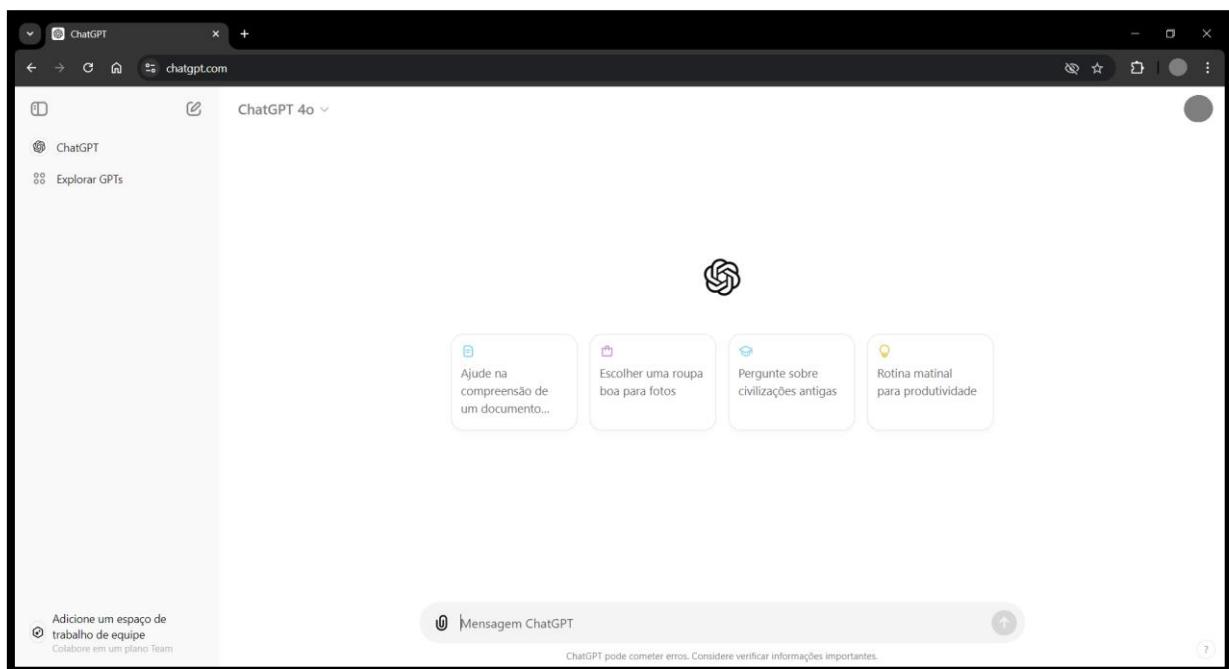
Fonte: Elaborado pelo autor.

<sup>1</sup> Disponível em: <<https://www.st.com/en/evaluation-tools/nucleo-g071rb.html>>. Acessível em: 07 set. 2024

### 3.2 Etapas do Desenvolvimento

As etapas do desenvolvimento dos experimentos foram baseadas na dinâmica de interação de usuários e da interface humano-computador com as ferramentas de IA Generativa disponíveis. Todas elas são baseadas em aplicações *online* realizadas por páginas Web em estilo conversacional apresentando um *prompt* ao usuário que permite fornecer ao sistema as entradas com o conteúdo da solicitação desejada. No caso do uso do ChatGPT e para os propósitos desse trabalho, as entradas foram feitas através de textos informados na interface padrão da aplicação. A Figura 5 mostra a tela do ChatGPT com a caixa de texto associada ao *prompt* de entrada.

Figura 5 – Tela principal do ChatGPT 4o.



Fonte: Página do ChatGPT 4o<sup>2</sup>.

A Figura 6 mostra as etapas realizadas para a produção dos resultados dos experimentos. O processo geral foi dividido basicamente em três etapas, cada qual com um propósito orientado à interação do usuário com a ferramenta de IA Generativa:

**1. Concepção:** Nesta etapa foram elaborados: (a) os textos das estórias de usuário que definiram os requisitos do produto; (b) os textos das especificações gerais, que consistiu no desritivo das características e restrições da solução a ser projetada e construída; e (c) os textos

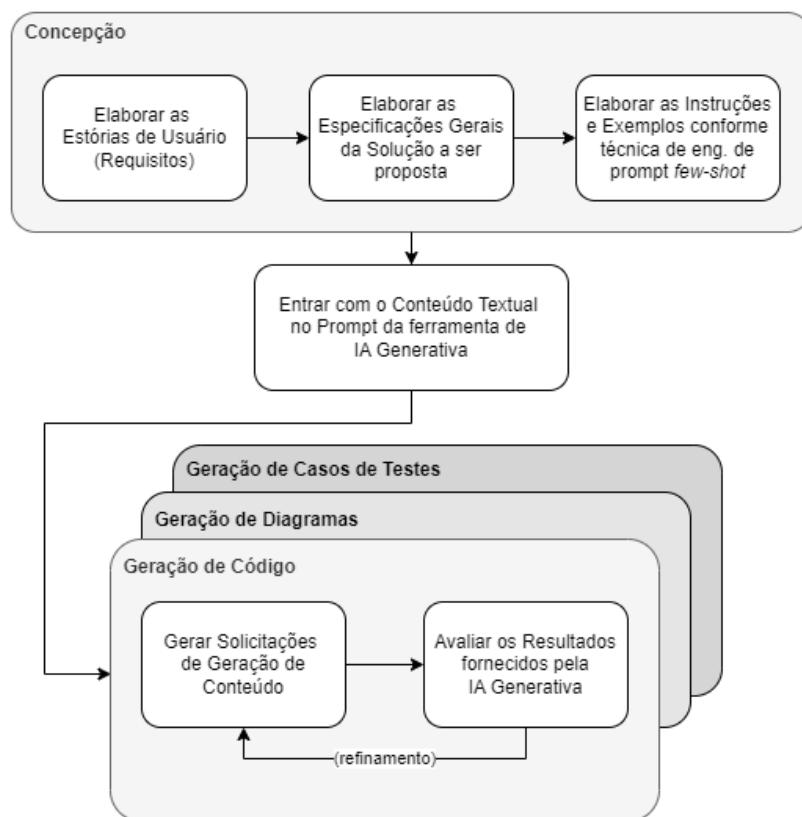
<sup>2</sup> Disponível em: <<https://chatgpt.com>>. Acesso em: 07 set. 2024.

das instruções e exemplos, quando aplicáveis, para melhorar a assertividade dos resultados gerados a partir das solicitações.

**2. Entrada de dados na ferramenta de IA Generativa:** Nesta etapa foi utilizado o mecanismo padrão de interação com o ChatGPT – a caixa de texto referente ao *prompt* da aplicação – para introduzir os textos elaborados na etapa 1 na ferramenta com o propósito de servirem como contexto para as solicitações posteriores. Importante destacar que, para aumentar a assertividade dos resultados e diminuir os ciclos de refinamento, as instruções e exemplos da etapa 1c foram baseados na técnica de engenharia de prompt *few-shot*.

**3. Geração dos Artefatos de Software:** Nesta etapa foram realizadas as solicitações para o ChatGPT gerar o conteúdo pretendido conforme o tipo de artefato de interesse: (a) geração de código-fonte na linguagem de programação desejada, (b) geração de diagramas UML para visualização da estrutura dos elementos e fluxos do código-fonte, e (c) geração de casos de teste para verificação do código-fonte gerado quando colocado em operação. Após a solicitações serem inseridas para processamento pelo ChatGPT, os resultados produzidos por ele foram avaliados quanto ao seu grau de corretude (compilar e funcionar corretamente) e completude (atender a todas as funcionalidades especificadas).

Figura 6 – As etapas e o fluxo básico para realização dos experimentos.



Fonte: Elaborado pelo autor.

### 3.3 Projeto do Dispositivo Eletrônico

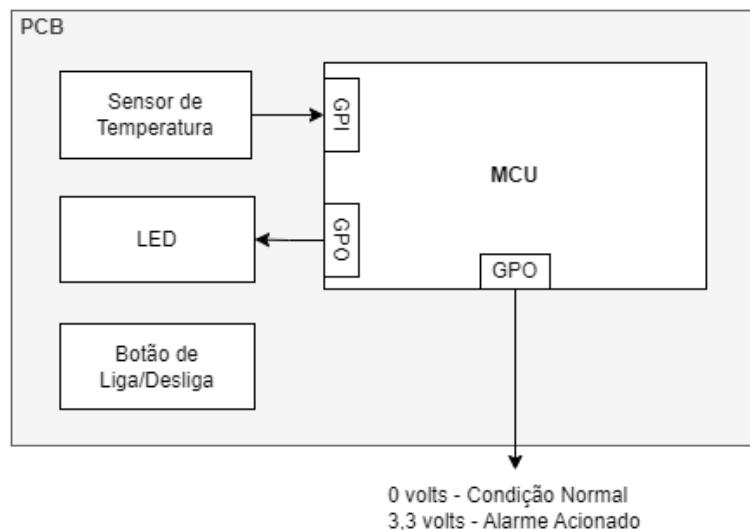
O projeto hipotético proposto para avaliação dos resultados dos artefatos gerados pelo ChatGPT nesse trabalho foi um dispositivo eletrônico que monitora a temperatura de interesse e dispara um alarme caso o valor da temperatura exceda um valor previamente definido.

A leitura do valor da temperatura é feita por um pino de entrada analógico associado a um conversor analógico-digital (ADC) interno ao microcontrolador com resolução de 12 bits. Isso significa que os valores lidos em um intervalo de tensão de 0 a 3,3 volts são representados por valores inteiros no intervalo de 0 a  $2^{12} - 1$ , o que representa valores inteiros de 0 a 4095.

O disparo do alarme consiste na sinalização luminosa por um LED piscando em frequência fixa de 4 Hertz e o acionamento de uma saída digital cujo nível lógico 0 (zero) refere-se a tensão de 0 volts (ausência de tensão) e o nível lógico 1 (um) refere-se a tensão de 3,3 volts presente na saída digital. Esse sinal pode ser utilizado para acionamento de outro sistema eletrônico acoplado ao dispositivo em desenvolvimento. A Figura 7 mostra o diagrama de blocos contendo os elementos do dispositivo eletrônico de projeto hipotético do experimento.

Para fins de simplificação, não foram tratados nos experimentos a conversão da escala de temperatura do sensor ao intervalo de tensão do ADC para determinação da temperatura efetiva a ser monitorada. Ao invés disso, optou-se por uma especificação de controle baseada no valor de tensão lido diretamente pelo ADC do microcontrolador. Assumiu-se que o escopo do experimento trata exclusivamente do potencial de geração de código para da lógica de controle necessária para o acionamento do LED e da saída de alarme.

Figura 7 – Diagrama de blocos simplificado da solução pretendida



Fonte: Elaborado pelo autor.

### 3.4 Estórias de Usuário e Especificações Gerais

Para a especificação dos requisitos da solução pretendida foram criadas estórias de usuário e especificações gerais com o objetivo de se determinar o funcionamento da lógica de controle do monitoramento de temperatura a ser realizado pelo dispositivo eletrônico. Tais artefatos, conforme o processo, foram elaborados nas etapas 1a e 1b para serem posteriormente informados ao ChatGPT como contexto para as solicitações de geração de código. A Tabela 2 mostra as estórias de usuário elaboradas e a Tabela 3 traz as especificações gerais para apoio à geração de código direcionado à placa de desenvolvimento utilizada.

Tabela 2 – Estórias de usuário utilizadas no experimento.

Identificação	Descriutivo
EU1	Como um usuário do dispositivo, eu quero definir um [valor limite de temperatura] que deve ser armazenado no dispositivo para a realização do [monitoramento de temperatura].
EU2	Como um usuário do dispositivo, eu quero poder ligar o dispositivo e este iniciar o [monitoramento da temperatura] realizando a leitura do [valor atual da temperatura] de modo contínuo e sem interrupção até que seja desligado.
EU3	Como um usuário do dispositivo, eu quero que o dispositivo mantenha a [saída digital de alarme] no nível lógico zero caso o [valor limite de temperatura] não tenha sido excedido.
EU4	Como um usuário do dispositivo, eu quero que o dispositivo mantenha a [saída digital de alarme] no nível lógico um caso o [valor limite de temperatura] tenha sido excedido.
EU5	Como um usuário do dispositivo, eu quero também que o dispositivo realize uma sinalização luminosa através da piscagem [LED de alarme] quando o [valor limite da temperatura] exceder o [valor atual da temperatura].

Fonte: Elaborado pelo autor.

Tabela 3 – Especificações gerais para operação na placa de desenvolvimento.

Identificação	Descriutivo
EG1	O [valor limite de temperatura] deve ser um valor entre 0 e 250 graus Celsius.
EG2	O [valor atual da temperatura] deve ser lido através da conversão analógico digital utilizando o ADC0 canal 0 definido no pino PA0.
EG3	O [LED de alarme] deve ser implementado através do pino PA5.
EG4	A [saída digital de alarme] deve ser implementado no pino PA6.
EG5	A piscagem do [LED de alarme] deve ocorrer a uma frequência de 4 Hertz.

Fonte: Elaborado pelo autor.

A técnica de engenharia de *prompt* de poucos disparos (*few-shot*) sugere que sejam fornecidas demonstrações no *prompt* para direcionar o modelo a ter um melhor desempenho e, portanto, conseguir trazer resultados mais assertivos. Isso porque, embora os modelos de linguagem de grande escala demonstrem resultados notáveis quando usando zero disparo (*zero-shot*), eles ainda ficam aquém em tarefas mais complexas.

As demonstrações providas para o ChatGPT considerando os objetivos desse experimento foram determinações sobre como realizar a geração de código que é solicitada nas interações. Tais demonstrações visaram fornecer exemplos de como codificar variáveis, funções e realizar a separação de código para facilitar a avaliação, além de melhorar a qualidade dos resultados de um modo geral. A Tabela 4 mostra as demonstrações, orientações e exemplos fornecidos ao ChatGPT para os experimentos realizados.

Tabela 4 – Demonstrações fornecidas ao ChatGPT com base em prompts *few-shot*.

Identificação	Descriutivo
DE1	<p>Nomes de variáveis globais de escopo de arquivo devem seguir o padrão PascalCase prefixadas com letra ‘m’ e precedidas com o operador ‘static’.</p> <p>Exemplo de nomes de variáveis globais de escopo de arquivo:</p> <pre>- float mCurrentTemperature = 0; - uint32_t mLedBlinkInterval = 125;</pre>
DE2	<p>Nomes de variáveis locais devem seguir o padrão camelCase.</p> <p>Exemplo de nomes de variáveis locais:</p> <pre>- float currentTemperature = 50; - uint32_t counter = 0;</pre>
DE3	<p>Nomes de funções de escopo de arquivo devem seguir o padrão camelCase e precedidas com o operador ‘static’.</p> <p>Exemplo de nomes de funções de escopo de arquivo:</p> <pre>- static float readTemperature(void); - static void checkAlarmCondition(void);</pre>
DE4	<p>Nomes de funções de escopo global devem seguir o padrão PascalCase e precedidas pelo substantivo separada por sublinhado pela ação com a ação seguindo o padrão camelCase.</p> <p>Exemplo de nomes de funções de escopo global:</p> <pre>- void Led_turnOn(void); - void Alarm_trigger(void);</pre>
DE5	<p>Os nomes dos arquivos gerados devem seguir o padrão snake_case.</p> <p>Exemplos de nomes de arquivos no padrão snake_case:</p> <pre>- led.h led.c - alarm_monitor.h alarm_monitor.c</pre>

Fonte: Elaborado pelo autor.

## 4 RESULTADOS

Após elaborados os artefatos previstos nas etapas 1a, 1b e 1c, todo esse conteúdo textual foi submetido ao *prompt* do ChatGPT no formato de texto simples derivado das tabelas 2, 3 e 4 da seção anterior. Esse conteúdo textual prévio permitiu que o modelo do ChatGPT entendesse o contexto da interação. Tal fato pôde ser observado nas respostas que foram geradas, nas Tabelas 5, 6, 7 e 8, a partir da entrada desses textos sem solicitação de ação explícita. O resultado na íntegra dessas interações iniciais pode ser consultado no Anexo A.

Tabela 5 – Preparação da interação com o ChatGPT com base dos artefatos da concepção.

Interação	Conteúdo
Solicitação	Vamos elaborar uma solução de um dispositivo eletrônico para controle de temperatura e disparo de alarmes. Os requisitos serão dados a seguir.
Resposta do ChatGPT	<i>Ótimo! Vamos detalhar a solução com base nos requisitos. Quais são os requisitos para o dispositivo de controle de temperatura? Se já tiver alguns em mente, podemos começar a partir daí.</i>

Fonte: Elaborado pelo autor.

Tabela 6 – Interação com ChatGPT para a entrada das estórias de usuário.

Interação	Conteúdo
Solicitação	Estes são os requisitos: (copiado o texto da Tabela 2 no prompt). Aguarde que irei informá-lo sobre algumas especificações adicionais a seguir.
Resposta do ChatGPT	<i>Entendido! Temos os requisitos básicos para o controle de temperatura e alarmes, incluindo o armazenamento de um valor limite de temperatura, a leitura contínua do valor atual da temperatura, controle de uma saída digital de alarme e sinalização por LED. Fico no aguardo das especificações adicionais para que possamos elaborar a solução completa.</i>

Fonte: Elaborado pelo autor.

Tabela 7 – Interação com ChatGPT para a entrada das especificações gerais.

Interação	Conteúdo
Solicitação	Estas são as especificações adicionais: (copiado o texto da Tabela 3 no prompt).

---

Aguarde que irei informá-lo sobre algumas orientações e exemplos de como iremos gerar código-fonte para esta solução.

Resposta do ChatGPT	<i>Perfeito! Com essas especificações, já podemos visualizar boa parte do design do sistema. Ficarei aguardando as orientações e exemplos sobre o código-fonte para implementar essa solução.</i>
---------------------	---

---

Fonte: Elaborado pelo autor.

Tabela 8 – Interação com ChatGPT para a entrada das orientações e exemplos (*few-shot*).

<b>Interação</b>	<b>Conteúdo</b>
Solicitação	<p>Estas são as orientações e exemplos sobre o código-fonte: (copiado o texto da Tabela 4 no prompt).</p> <p>Depois de entender essas orientações, irei realizar uma solicitação a seguir para geração de código-fonte em uma linguagem específica.</p>
Resposta do ChatGPT	<i>Ótimo! As orientações e exemplos sobre os padrões de nomenclatura e estrutura de código estão bem claras. Estou pronto para a próxima solicitação de geração de código-fonte na linguagem específica que você escolher.</i>

Fonte: Elaborado pelo autor.

A partir daí, na mesma janela de chat cujo conteúdo textual das estórias de usuário, das especificações gerais e das demonstrações e exemplos previamente informados ao ChatGPT, foram geradas solicitações de ação no *prompt* para a geração dos conteúdos gerais de interesse.

#### 4.1 Resultados da Geração de Código-fonte

A primeira solicitação de interesse, mostrada na Tabela 9, foi a geração de código-fonte em linguagem C baseado na utilização da biblioteca HAL fornecida pelo fabricante do microcontrolador aos desenvolvedores. A utilização dessa biblioteca é recomendada porque abstrai dos desenvolvedores o controle e operação dos registradores internos do microcontrolador e permite que a lógica de firmware a ser desenvolvida seja totalmente focada na aplicação final da solução. O amplo e profundo conhecimento dessa biblioteca é um desafio para desenvolvedores menos experientes quando expostos a terem que interagir com todos os detalhes dos sistemas internos, dos periféricos e das interfaces de comunicação do microcontrolador. Aqui já se esperou o apoio da IA Generativa no conhecimento sobre essas bibliotecas especializadas visando aumentar a produtividade da produção de código.

Tabela 9 – Solicitação e resposta para geração de código-fonte em linguagem C.

Interação	Conteúdo
Solicitação	Gere o código-fonte baseado em boas práticas de engenharia de software para implementar os requisitos, funcionalidades e especificações adicionais informados na linguagem de programação C utilizando a biblioteca HAL da ST e considerando o microcontrolador STM32G071.
Resposta do ChatGPT	<i>Aqui está a implementação do código-fonte em C para o controle de temperatura e alarme, usando a biblioteca HAL da ST para o microcontrolador STM32G071, considerando as boas práticas de engenharia de software e os requisitos fornecidos:</i>

(código-fonte completo pode ser consultado no Anexo B).

**Explicação:**

- **Leitura da temperatura:** A função `readTemperature()` lê o valor da temperatura do sensor conectado ao ADC (PA0). A conversão de ADC para temperatura é simplificada para 0-250°C.
- **Verificação de alarme:** A função `checkAlarmCondition()` compara o valor atual da temperatura com o limite e aciona o alarme e o LED se necessário.
- **Controle do LED e alarme:** Funções específicas para ligar, desligar e piscar o LED (PA5) e ativar ou resetar o alarme na saída digital (PA6).
- **Estrutura e organização:** O código está modularizado, com responsabilidades claras para cada função e arquivo.

*Caso precise de ajustes ou mais detalhes, estou à disposição!*

---

Fonte: Elaborado pelo autor.

A Tabela 10 mostra a lista dos arquivos de código-fonte gerados com base na solicitação realizada da Tabela 9. Foi possível notar que a orientação DE5 da Tabela 4 foi devidamente observada para a geração dos nomes de arquivos.

Tabela 10 – Descritivo da geração de código-fonte conforme a solicitação.

Arquivos	Conteúdo
<code>led.h</code> <code>led.c</code>	Arquivos contendo as definições e implementações das funções do LED documentadas no padrão Doxygen.  <code>void Led_turnOn(void);</code> <code>void Led_turnOff(void);</code> <code>void Led_blink(uint32_t interval);</code>

alarm_monitor.h alarm_monitor.c	Arquivos contendo as definições e implementações das funções do monitoramento do alarme documentadas no padrão Doxygen.  void Alarm_trigger(void); void Alarm_reset(void);
main.h main.c	Arquivos contendo as definições e implementações das funções gerais de inicialização e operação do laço principal ( <i>super loop</i> ) do firmware.  void SystemClock_Config(void); void MX_GPIO_Init(void); void MX_ADC1_Init(void);

---

Fonte: Elaborado pelo autor.

Analizando-se o resultado produzido sob a perspectiva do atendimento às orientações e restrições informadas previamente, foi possível observar que o ChatGPT gerou os artefatos com plena compatibilidade, atentando-se aos detalhes do padrão e estilo de codificação variados que estavam associados aos nomes de arquivos, funções e variáveis a serem gerados.

Analizando-se o resultado produzido sob a perspectiva do funcionamento do código quando executado na placa de desenvolvimento, foi possível observar que atendeu quase que na sua totalidade as estórias de usuário especificadas, só deixando de considerar o requisito particular referente à piscagem do LED (EG5). A primeira implementação resultante da primeira interação, portanto, conteve um defeito no controle do período de piscagem do LED.

Para resolver esse defeito foi realizada uma nova solicitação ao ChatGPT conforme a Tabela 11 informando-o do problema detectado para verificar a sua capacidade de resolução.

Tabela 11 – 1<sup>a</sup> solicitação para resolução de defeito em código-fonte gerado pelo ChatGPT.

Interação	Conteúdo
Solicitação	Aprimore a função Led_blink para contemplar a verificação do intervalo de piscagem que foi definido pela variável mLedBlinkInterval. Da forma como foi gerado, a função não observa o intervalo de tempo e a implementação da solução está piscando a cada 100 ms, que é tempo da pequena espera que foi inserido para suavizar. Não é este tempo que deve ser observado. O tempo que deve ser atendido é o da variável mLedBlinkInterval.
Resposta do ChatGPT	<i>Para aprimorar a função Led_blink para que observe corretamente o intervalo de tempo definido pela variável mLedBlinkInterval, precisamos garantir que a função leve em consideração o tempo que se passou desde a última alteração do estado do LED e somente execute a mudança de estado quando o tempo apropriado tiver transcorrido.</i> <i>A abordagem é usar uma variável estática para armazenar o "timestamp" da última vez que o LED foi alternado. O HAL_GetTick() da HAL pode ser</i>

*utilizado para obter o tempo decorrido desde o início do programa, em milissegundos, o que permite controlar o intervalo corretamente.*

*Aqui está o código aprimorado para a função Led\_blink:*

(código-fonte completo pode ser consultado no Anexo C).

*Com essa alteração, o LED piscará corretamente conforme o intervalo definido por mLedBlinkInterval, independentemente do tempo de execução do loop principal, que possui um delay de 100 ms.*

*Essa implementação garante que o LED pisque exatamente a cada 125 ms (ou 4 Hz) quando necessário.*

*Caso tenha mais alguma solicitação, fico à disposição!*

Fonte: Elaborado pelo autor.

Analizando-se o resultado elaborado após a solicitação de correção, notou-se que código produzido agora continha a verificação do intervalo de tempo da piscagem do LED na lógica da função de piscagem, mas ainda não fazia com que o LED piscasse na frequência desejada, mesmo com a afirmação do ChatGPT (no trecho final do diálogo) dizendo que a implementação garantia a piscagem do LED no intervalo de tempo especificado.

O defeito foi o tempo de espera que ele gerou e incluiu no laço principal, não observando o fato que esse tempo de espera define um ciclo de 100 milissegundos para execução da lógica de verificação da condição de alarme. O LED deve perfazer um ciclo completo de intervalo de 250 milissegundos (4 Hertz), sendo um tempo de 125 milissegundos para cada inversão do estado do LED entre ligado e desligado. Esses intervalos de tempo não são múltiplos de 100 milissegundos do laço principal, o que faz com que a ação de inversão do estado do LED não seja realizada de modo a cumprir a frequência de 4 Hertz conforme era esperado.

Novamente o ChatGPT foi informado sobre o defeito e uma nova solicitação de correção do defeito foi feita, conforme mostra a Tabela 12.

Tabela 12 – 2<sup>a</sup> solicitação para resolução de defeito em código-fonte gerado pelo ChatGPT.

Interação	Conteúdo
Solicitação	Ainda não é possível ter o LED piscando corretamente porque o delay de 100 ms no loop principal não é divisor do intervalo de piscagem do LED. Por favor, corrija.
Resposta do ChatGPT	<i>Entendi! O problema ocorre porque o delay de 100 ms no loop principal interfere no tempo de piscagem do LED, pois o valor de mLedBlinkInterval (125 ms) não é um divisor exato de 100 ms. Para corrigir isso, precisamos</i>

*evitar que o delay do loop principal controle o tempo de piscagem do LED. A solução é separar o controle do LED da lógica do loop principal.*

*Uma maneira eficaz de resolver esse problema é refinar a lógica para que a função de piscagem do LED não dependa de nenhum delay explícito no loop principal. Em vez disso, podemos verificar constantemente o tempo usando HAL\_GetTick() para que a função de piscagem atue de maneira independente.*

*Aqui está a solução corrigida:*

(código-fonte completo pode ser consultado no Anexo D).

*Com isso, a função Led\_blink() agora garante que o LED piscará exatamente a cada 125 ms, respeitando o intervalo de mLedBlinkInterval, mesmo sem delays adicionais no loop principal.*

*Essa abordagem é mais precisa e evita qualquer interferência entre o controle da piscagem do LED e o tempo de execução do loop principal. Caso tenha mais alguma dúvida ou ajuste, estou à disposição!*

Fonte: Elaborado pelo autor.

Dessa vez o código produzido atendeu o requisito faltante preservando o funcionamento da implementação que atendia aos demais requisitos. A resolução foi anular o tempo de espera do laço principal e manter a lógica de contagem do intervalo de tempo na função de piscagem do LED, que agora passou a contabilizar o tempo corretamente dado que a chamada a essa função ocorre bem mais frequentemente (frequência de operação baseada no *clock* do processador) que o tempo de intervalo definido para atender ao requisito dos 4 Hertz.

Essa abordagem, porém, poderia ser melhorada com uso de temporizadores internos (*timers*) disponíveis no microcontrolador em questão, que operam de modo independente da lógica principal executada pelo núcleo do processador presente no microcontrolador. O uso de temporizadores internos não estava previsto nos requisitos e, portanto, não foi considerada essa possibilidade pelo ChatGPT.

Essa extração de conhecimentos para sugerir melhoria de tal natureza não ocorreu no ChatGPT e usualmente corrobora a ideia de que a IA Generativa, no geral, é mais uma ferramenta para geração de conteúdo e manipulação de símbolos de linguagens a ser realmente uma superinteligência capaz de gerar *insights* não baseados exclusivamente em textos.

Dessa forma, para que o código-fonte final agora pudesse contemplar tal melhoria, foi feita uma nova solicitação tendo o requisito explícito para o uso de temporizador presente no microcontrolador. Essa interação pode ser vista na Tabela 13.

Tabela 13 – Solicitação para melhoria no código-fonte gerado pelo ChatGPT.

Interação	Conteúdo
Solicitação	Vamos refatorar o código gerado anteriormente para que o controle de piscagem do LED seja feito por um timer interno do microcontrolador que é ativado quando o alarme é disparado e desativado caso contrário. Podemos usar o TIM1 para este propósito. No laço principal insira uma nova variável para definir o tempo de espera o ciclo de verificação da condição de alarme como sendo de 50 ms. Este tempo poderá ser redefinido posteriormente.
Resposta do ChatGPT	<p><i>Aqui está a refatoração do código para utilizar o <b>TIM1</b> para controlar a piscagem do LED e a nova variável que define o tempo de espera do ciclo de verificação da condição de alarme:</i></p> <p>(código-fonte completo pode ser consultado no Anexo E).</p> <p><i>Com essa solução, o LED de alarme pisca a cada 125 ms (4 Hz) enquanto o alarme está ativo, controlado pelo <b>TIM1</b>, de forma independente do loop principal. Caso precise de mais ajustes ou esclarecimentos, estou à disposição!</i></p>

Fonte: Elaborado pelo autor.

O código-fonte resultado da implementação da melhoria com a inclusão de temporizadores deu origem a dois novos arquivos de código-fonte: `tim.h` e `tim.c`, referentes às funções dos temporizadores. Consultando tais arquivos, notou-se que observaram o atendimento às orientações e demonstrações previstas na Tabela 4.

Como pode ser visto, as etapas previstas no fluxo da Figura 6 preveem ciclos iterativos de solicitação e avaliação dos resultados de modo a serem aplicados refinamentos até que o resultado desejado seja produzido. Para o caso desse experimento de geração de código, foram necessárias três iterações adicionais posteriores à solicitação inicial, sendo duas delas para correção de defeitos e uma para implementação de melhoria.

Como experimento final na geração de código-fonte, considerou-se a possibilidade de se refinar os requisitos adicionando uma nova especificação (EG6) para indicar que a piscagem do LED devia ser construída usando temporizador. O objetivo foi apurar se o código-fonte gerado apresentava os mesmos defeitos ou eventualmente outros defeitos similares. A Tabela 14 mostra a atualização da Tabela 3 adicionando-se a nova especificação para geração de código considerando o temporizador interno.

Tabela 14 – Atualização das especificações gerais para melhoria da geração de código.

Identificação	Descritivo
EG1	O [valor limite de temperatura] deve ser um valor entre 0 e 250 graus Celsius.
EG2	O [valor atual da temperatura] deve ser lido através da conversão analógico digital utilizando o ADC0 canal 0 definido no pino PA0.
EG3	O [LED de alarme] deve ser implementado através do pino PA5.
EG4	A [saída digital de alarme] deve ser implementado no pino PA6.
EG5	A piscagem do [LED de alarme] deve ocorrer a uma frequência de 4 Hertz.
EG6	A piscagem do [LED de alarme] deve ocorrer por meio do timer TIM1.

Fonte: Elaborado pelo autor.

Para a condução do experimento agora considerando EG6, foi iniciado um novo chat no ChatGPT e refeitas as mesmas interações descritas nas Tabelas de 5 a 9, visando garantir que as interações prévias não tivessem influência nos resultados dessa nova rodada.

Dessa vez, o ChatGPT produziu um código-fonte funcional (verificado na placa de desenvolvimento), livre de defeitos e que atendeu a todos os requisitos da especificação e as funcionalidades previstas nas estórias de usuário. A Tabela 15 e a Tabela 16 mostram o resultado da interação e a estrutura do novo código-fonte resultante, respectivamente.

Tabela 15 – Solicitação para geração de código-fonte com a especificação atualizada.

Interação	Conteúdo
Solicitação	Gere o código-fonte baseado em boas práticas de engenharia de software para implementar os requisitos, funcionalidades e especificações adicionais informados na linguagem de programação C utilizando a biblioteca HAL da ST e considerando o microcontrolador STM32G071.
Resposta do ChatGPT	<i>Aqui está a implementação em C usando a biblioteca HAL para o microcontrolador STM32G071, baseada nos requisitos e especificações fornecidas. Para organização, os arquivos seguem o padrão snake_case, conforme suas orientações.</i>

(código-fonte completo pode ser consultado no Anexo F).

### Resumo

- **Pinos utilizados:**
  - ADC para leitura de temperatura no PA0.
  - LED no PA5.
  - Saída de alarme no PA6.
- **Funcionamento:**

- *O dispositivo lê continuamente a temperatura.*
- *Se a temperatura ultrapassar o limite, o alarme é ativado e o LED pisca a 4 Hz.*
- *A piscagem do LED é controlada por um timer interno (TIM1).*

*Este código foi desenvolvido com boas práticas de engenharia de software, organizando as funcionalidades em módulos e seguindo as convenções estabelecidas.*

Fonte: Elaborado pelo autor.

Tabela 16 – Descritivo da geração de código-fonte atualizada conforme a solicitação.

Arquivos	Conteúdo
led.h led.c	Arquivos contendo as definições e implementações das funções do LED documentadas no padrão Doxygen.  <code>void Led_init(void); void Led_startBlink(void); void Led_stopBlink(void);</code>
temperature_sensor.h temperature_sensor.c	Arquivos contendo as definições e implementações das funções do sensor de temperatura documentadas no padrão Doxygen.  <code>void TemperatureSensor_init(void); float TemperatureSensor_read(void);</code>
alarm_monitor.h alarm_monitor.c	Arquivos contendo as definições e implementações das funções do monitoramento do alarme documentadas no padrão Doxygen.  <code>void Alarm_init(void); void Alarm_checkCondition(float currentTemperature,                            float temperatureLimit);</code>
main.c	Arquivos contendo a implementação das funções gerais de inicialização e operação do laço principal ( <i>super loop</i> ) do firmware.  <code>void SystemClock_Config(void); int main(void);</code>

Fonte: Elaborado pelo autor.

A partir desses resultados, não houve mais necessidade de refinamentos e a avaliação foi a de que o código-fonte gerado atendeu aos requisitos e produziu a solução pretendida. É importante salientar que os arquivos gerados foram necessários, mas não suficientes para se produzir a solução completa. Para a compilação e execução desse código na placa de desenvolvimento foi necessário criar um projeto na ferramenta STM32CubeIDE e informar o número (*Part Number*) da placa de desenvolvimento. A partir daí a ferramenta criou uma estrutura de projeto com código-fonte padrão e, então, o código gerado pelo ChatGPT foi inserido para compor o firmware final que é executado na placa de desenvolvimento.

## 4.2 Resultados da Geração de Diagramas

Com a solução pretendida disponível em código-fonte gerado pelo ChatGPT, o próximo experimento foi ter uma documentação adicional do firmware na forma de diagramas em notação de linguagem visual, seja para se ter uma visualização gráfica da sua estrutura – o que pode vir a implicar novos refinamentos para favorecer algum estilo arquitetural que se deseje – , seja para fins de arquivamento junto ao código-fonte do projeto para posterior manutenção por outros desenvolvedores.

Esse experimento teve como objetivo verificar as capacidades do ChatGPT em gerar diagramas a partir do código-fonte gerado anteriormente. Para viabilizar o controle de versão em projetos também do diagrama além do próprio código-fonte, foi solicitado que o ChatGPT gerasse a representação gráfica do código em um diagrama UML usando a notação textual definida pela ferramenta PlantUML.

Como os LLM's são especializadas em geração de conteúdo textual, a tarefa em questão nesse experimento foi a transformação do código-fonte escrito em linguagem de programação C para a linguagem de notação textual da ferramenta PlantUML. O que se observou foi que o ChatGPT foi capaz de gerar o conteúdo de texto na notação solicitada. Este conteúdo pôde ser usado para elaboração e visualização do diagrama diretamente no site da ferramenta online PlantUML.

A Tabela 17 mostra a solicitação feita ao ChatGPT para geração do diagrama em notação textual e a Figura 8 mostra o resultado da visualização gráfica do diagrama de classes obtido através do processamento do resultado na ferramenta online PlantUML.

Tabela 17 – Solicitação e resposta para geração de diagrama de classes feita ao ChatGPT.

Interação	Conteúdo
Solicitação	Vamos gerar uma representação gráfica através de um diagrama de classes da UML usando a notação para uso na ferramenta PlantUML.
Resposta do ChatGPT	<i>Aqui está a notação do diagrama de classes em PlantUML para que você possa usá-la em qualquer ferramenta compatível com o PlantUML:</i>

```

@startuml

class main {
    - TEMPERATURE_LIMIT: float
    + main(): void
    - SystemClock_Config(): void

```

```

}

class AlarmMonitor {
    + Alarm_init(): void
    + Alarm_checkCondition(currentTemperature: float,
                           temperatureLimit: float): void
    - alarmTrigger(): void
    - alarmReset(): void
}

class Led {
    + Led_init(): void
    + Led_startBlink(): void
    + Led_stopBlink(): void
    - ledBlinkConfig(): void
}

class TemperatureSensor {
    + TemperatureSensor_init(): void
    + TemperatureSensor_read(): float
}

main --> AlarmMonitor : uses
main --> Led : uses
main --> TemperatureSensor : uses

@enduml

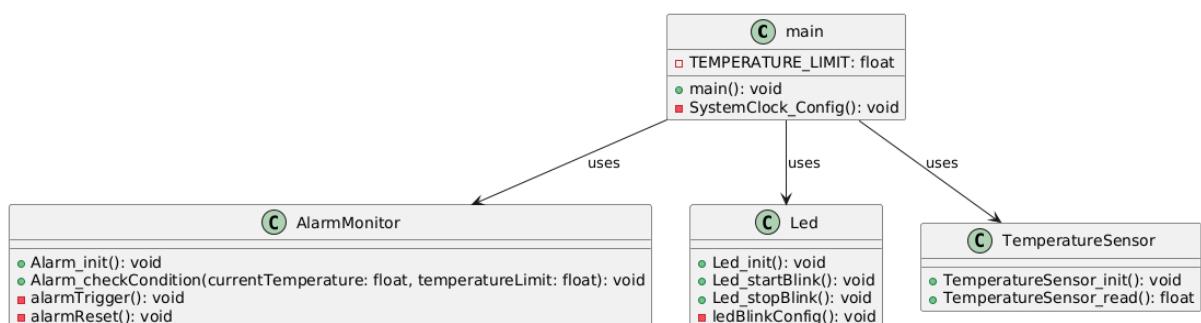
```

*Basta copiar esse código e colá-lo em um ambiente PlantUML para gerar o diagrama.*

---

Fonte: Elaborado pelo autor.

Figura 8 – Diagrama de classes da UML gerado pelo ChatGPT



Fonte: Elaborado pelo autor<sup>3</sup>.

---

<sup>3</sup> Elaborado com uso da ferramenta online PlantUML.  
Disponível em <<https://www.plantuml.com/>>. Acesso em 07 set. 2024.

Assim como a estrutura do programa pôde ser visualizada através de um diagrama, seu fluxo de operação também pôde ser representado graficamente. A UML define alguns diagramas para visualização de fluxos, tais como o diagrama de atividades e o diagrama de sequência. Como se pretendeu visualizar a operação interna do código, o diagrama de sequência foi escolhido. A Tabela 18 mostra a solicitação feita ao ChatGPT para geração do diagrama em notação textual e a Figura 9 mostra o resultado da visualização gráfica do diagrama obtido através do processamento do resultado na ferramenta online PlantUML.

Tabela 18 – Solicitação e resposta para geração de diagrama de sequência feita ao ChatGPT

Interação	Conteúdo
Solicitação	Vamos gerar agora um diagrama de sequência da UML a partir do código-fonte gerado anteriormente para que seja possível se visualizar o fluxo de operação do programa.
Resposta do ChatGPT	<p>Aqui está a notação em PlantUML para o diagrama de sequência, representando o fluxo de operação do programa com base no código-fonte gerado:</p> <pre> @startuml @startuml actor User  participant "main" as Main participant "AlarmMonitor" as AlarmMonitor participant "Led" as Led participant "TemperatureSensor" as TemperatureSensor  User -&gt; Main: Ligar o dispositivo Main -&gt; Led: Led_init() Main -&gt; AlarmMonitor: Alarm_init() Main -&gt; TemperatureSensor: TemperatureSensor_init()  loop [Continuo]     Main -&gt; TemperatureSensor: TemperatureSensor_read()     TemperatureSensor --&gt; Main: currentTemperature      Main -&gt; AlarmMonitor: Alarm_checkCondition(         currentTemperature, TEMPERATURE_LIMIT)      alt [Se temperatura excede o limite]         AlarmMonitor -&gt; Led: Led_startBlink()         AlarmMonitor -&gt; Main: Aciona alarme     else [Se temperatura não excede o limite]         AlarmMonitor -&gt; Led: Led_stopBlink()         AlarmMonitor -&gt; Main: Reseta alarme     end </pre>

```

Main -> Main: HAL_Delay(50)
end
@enduml

```

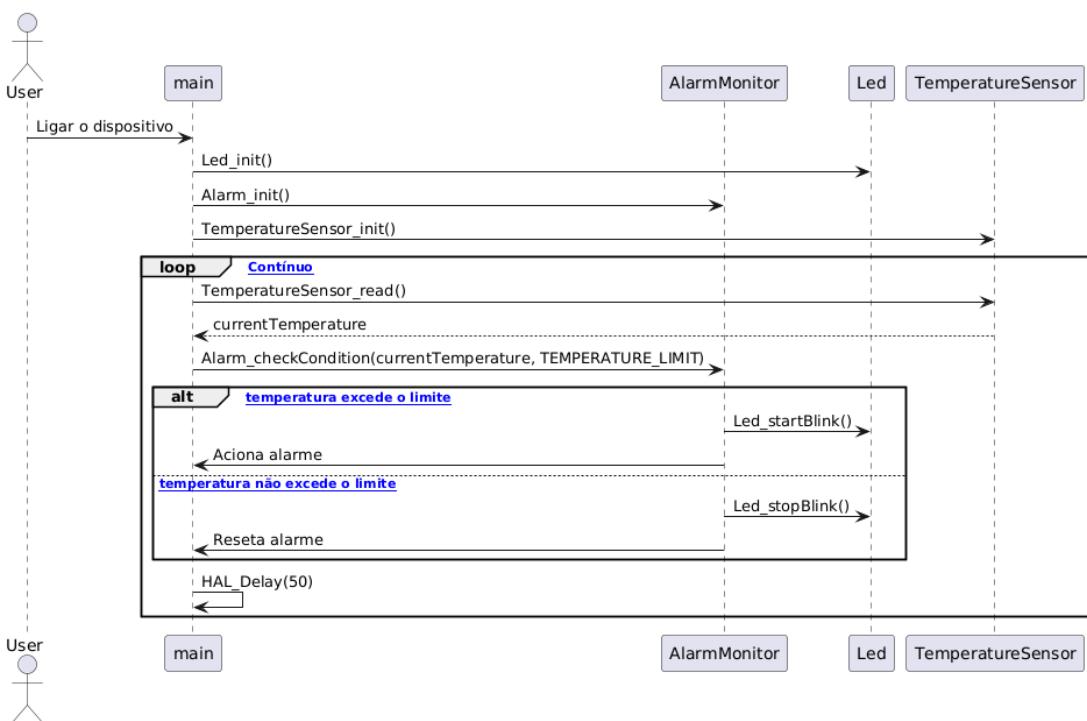
**Explicação:**

1. **User:** O usuário inicia o dispositivo.
2. **Main:** Inicializa o LED, alarme e sensor de temperatura.
3. **TemperatureSensor:** Lê a temperatura atual continuamente.
4. **AlarmMonitor:** Verifica se a temperatura excede o limite e aciona ou reseta o alarme.
5. **Led:** Pisca o LED de alarme quando a temperatura excede o limite.

Copie esse código para um ambiente compatível com PlantUML para gerar o diagrama de sequência.

Fonte: Elaborado pelo autor.

Figura 9 – Diagrama de sequência da UML gerado pelo ChatGPT



Fonte: Elaborado pelo autor<sup>4</sup>.

As visualizações gráficas através de diagramas gerados foram satisfatórias. A vantagem nessa abordagem é que a geração pode ser sempre realizada de forma automática pela ferramenta não havendo trabalho adicional para se manter a documentação atualizada. A desatualização da documentação é um aspecto de ocorrência frequente em projetos de software.

<sup>4</sup> Elaborado com uso da ferramenta online PlantUML.

Disponível em <<https://www.plantuml.com/>>. Acesso em 07 set. 2024.

### 4.3 Resultados da Geração de Casos de Teste

Da mesma forma que a geração de diagramas a partir de código-fonte, foi possível explorar as capacidades do ChatGPT relacionadas à transformação do código-fonte gerado em outra categoria de conteúdo textual bastante utilizada em projetos de desenvolvimento de software, que são os casos de teste.

Uma nova solicitação em sequência ao contexto corrente do chat utilizado para geração dos artefatos anteriores foi feita ao ChatGPT, agora para gerar cenários de teste possíveis para a solução que foi previamente desenvolvida. O resultado pode ser na Tabela 19.

Tabela 19 – Solicitação e resposta para geração de casos de teste feita ao ChatGPT.

Interação	Conteúdo
Solicitação	Agora vamos explorar a geração de casos de teste para a solução que foi desenvolvida. Gere os casos de teste em linguagem natural para execução manual por usuários interagindo com o dispositivo.
Resposta do ChatGPT	<i>Aqui estão alguns casos de teste em linguagem natural para execução manual por usuários interagindo com o dispositivo de controle de temperatura. Esses testes visam verificar se o dispositivo está funcionando corretamente de acordo com os requisitos e especificações definidos.</i>

#### Caso de Teste 1: Configuração do valor limite de temperatura

**Objetivo:** Verificar se o usuário consegue definir corretamente o valor limite de temperatura no dispositivo.

- **Pré-condições:** O dispositivo está desligado.
- **Passos:**
  1. Ligue o dispositivo.
  2. Defina o valor limite de temperatura para 100°C (por exemplo).
  3. Observe se o dispositivo salva e armazena corretamente o valor configurado.
- **Resultado Esperado:** O dispositivo aceita o valor limite de temperatura, que fica armazenado até ser redefinido ou até que o dispositivo seja desligado.

#### Caso de Teste 2: Monitoramento contínuo da temperatura

**Objetivo:** Verificar se o dispositivo realiza o monitoramento contínuo da temperatura.

- **Pré-condições:** O dispositivo está ligado com um valor limite de temperatura configurado.
- **Passos:**
  1. Verifique se o dispositivo está ligado.

2. Monitore o valor atual da temperatura exibido ou lido através do canal ADC no pino PA0.
  3. Aumente e diminua a temperatura ambiente ao redor do sensor para verificar se ele está registrando as mudanças de temperatura.
- **Resultado Esperado:** O dispositivo lê continuamente a temperatura sem interrupção.

**Caso de Teste 3: Comportamento do alarme (temperatura abaixo do limite)**

**Objetivo:** Verificar o comportamento do alarme quando a temperatura está abaixo do limite configurado.

- **Pré-condições:** O dispositivo está ligado com o valor limite de 100°C e a temperatura ambiente está abaixo desse valor.
- **Passos:**
  1. Mantenha a temperatura do sensor abaixo de 100°C.
  2. Observe a saída digital de alarme no pino PA6.
- **Resultado Esperado:** A saída digital de alarme permanece em nível lógico zero e o LED no pino PA5 permanece apagado.

**Caso de Teste 4: Disparo do alarme (temperatura acima do limite)**

**Objetivo:** Verificar o comportamento do alarme quando a temperatura excede o limite configurado.

- **Pré-condições:** O dispositivo está ligado com o valor limite de 100°C e a temperatura ambiente está abaixo desse valor.
- **Passos:**
  1. Aumente a temperatura ao redor do sensor para exceder 100°C.
  2. Observe a saída digital de alarme no pino PA6 e o comportamento do LED no pino PA5.
- **Resultado Esperado:** A saída digital de alarme vai para nível lógico um, e o LED de alarme pisca a uma frequência de 4 Hz.

**Caso de Teste 5: Comportamento do alarme ao retornar para temperatura abaixo do limite**

**Objetivo:** Verificar o comportamento do alarme quando a temperatura retorna para um valor abaixo do limite após ter sido excedida.

- **Pré-condições:** O alarme foi disparado porque a temperatura ultrapassou o limite.
- **Passos:**
  1. Reduza a temperatura do sensor para um valor abaixo de 100°C.
  2. Observe o comportamento do alarme e do LED.
- **Resultado Esperado:** A saída digital de alarme retorna para nível lógico zero e o LED para de piscar.

### **Caso de Teste 6: Frequência de piscagem do LED**

**Objetivo:** Verificar se o LED de alarme piscá na frequência correta quando o alarme é acionado.

- **Pré-condições:** O dispositivo está ligado e a temperatura excede o limite configurado.
- **Passos:**
  1. Acione o alarme aumentando a temperatura acima de 100°C.
  2. Cronometre o tempo entre as piscadas do LED de alarme.
- **Resultado Esperado:** O LED de alarme deve piscar com uma frequência de 4 Hz (um ciclo de piscada a cada 250 ms).

### **Caso de Teste 7: Estabilidade do sistema ao longo do tempo**

**Objetivo:** Verificar a estabilidade do dispositivo e do alarme após longos períodos de operação.

- **Pré-condições:** O dispositivo está ligado com um valor limite de temperatura configurado.
- **Passos:**
  1. Deixe o dispositivo ligado por 12 horas.
  2. Durante esse tempo, monitore o comportamento do alarme e do LED à medida que a temperatura varia ao longo do tempo.
- **Resultado Esperado:** O dispositivo deve funcionar de forma estável, sem travamentos ou falhas, e deve acionar o alarme corretamente quando necessário.

Esses casos de teste permitem ao usuário verificar se o dispositivo está cumprindo todos os requisitos funcionais e técnicos especificados, além de assegurar que o comportamento do dispositivo seja consistente em diferentes cenários de operação.

---

Fonte: Elaborado pelo autor.

Foram gerados sete casos de teste no total sem fornecer nenhuma obrigatoriedade de cobertura e nem critérios de teste específicos a serem atendidos. Seis deles visaram a verificação das funcionalidades diretamente especificadas nos requisitos e nas estórias de usuário. Um deles especificamente referiu-se a um teste não-funcional de longa duração, sugerindo deixar o dispositivo ligado por doze horas em funcionamento para verificar o seu comportamento.

Certamente a disciplina de teste pode ser bem mais vastamente explorada no ChatGPT devido a sua característica fundamental diretamente associada à especialidade das ferramentas baseadas em LLM's, que é a geração de conteúdo textual, e sendo os roteiros de casos de teste um tipo desse conteúdo. Os resultados com um único *prompt* ao ChatGPT foram satisfatórios.

## 5 CONCLUSÃO

Esse trabalho abordou a aplicação da IA Generativa, especificamente o ChatGPT, na geração de artefatos de software no contexto do desenvolvimento de firmware, com o objetivo principal de avaliar a eficácia dessas ferramentas no auxílio à criação de código-fonte e outros artefatos de software. O principal propósito foi verificar se a IA Generativa poderia ser utilizada como uma ferramenta prática e produtiva no processo de desenvolvimento de firmware, reduzindo a carga de trabalho e aumentando a eficiência dos desenvolvedores. Para atingir esse objetivo, foi realizado um experimento utilizando técnicas de engenharia de *prompt*, que guiaram a interação com o ChatGPT para a geração mais assertiva de código estruturado e compatível com as necessidades específicas do firmware.

A metodologia adotada consistiu em estruturar o problema em requisitos de software, descritos por meio de estórias de usuário, especificações técnicas e orientações sobre boas práticas de desenvolvimento. Esses artefatos foram inseridos no ChatGPT por meio de *prompts* detalhados, que incluíram exemplos e demonstrações para guiar a IA na produção de código de alta qualidade. O experimento utilizou um cenário realista de desenvolvimento de firmware para um dispositivo eletrônico de monitoramento de temperatura, com o objetivo de gerar código que controlasse um alarme e um LED de sinalização quando determinados limites de temperatura fossem excedidos.

Os resultados obtidos demonstraram que o ChatGPT foi capaz de produzir código-fonte funcional, organizado e apresentando aspectos de qualidade, atendendo a todos os requisitos estabelecidos nos experimentos. Em várias interações, o código gerado necessitou de ajustes para corrigir pequenos defeitos e adequar-se aos padrões de qualidade esperados, como foi o caso da frequência de piscagem do LED, que não foi inicialmente configurada corretamente. Ao longo do processo, foram feitas solicitações para correção e refinamento da solução, e o ChatGPT respondeu de forma eficaz, ajustando o código conforme o necessário. No entanto, ficou evidente que a IA Generativa, embora eficiente em várias tarefas, não prescinde completamente da intervenção humana, especialmente quando se trata de ajustes finos e otimizações mais complexas.

A conclusão geral é que o ChatGPT, e provavelmente outras ferramentas de IA Generativa que estão disponíveis, tem um grande potencial para auxiliar no desenvolvimento de firmware e outras áreas de software, aumentando a produtividade e automatizando tarefas repetitivas. No entanto, ainda há limitações no que diz respeito à sua capacidade de prever

problemas e sugerir melhorias que não sejam explicitamente solicitadas. Isso reforça a importância da colaboração entre desenvolvedores e a IA, onde a *expertise* humana permanece essencial para garantir a assertividade, robustez e a precisão dos sistemas desenvolvidos.

Como sugestões de trabalhos futuros, podem ser realizadas novas investigações e aprimoramentos na integração de ferramentas de IA generativa no ciclo de desenvolvimento de software explorando e incorporando mais técnicas de engenharia de *prompt*. No âmbito da etapa de concepção, podem ser realizados novos trabalhos que tragam experimentos baseados em outras técnicas de captura e especificação de requisitos, tais como casos de uso, texto livre, e até mesmo especificação formal. Na perspectiva da geração de código-fonte, além da linguagem C, pode ser explorada a geração ou a transformação de código para a linguagem C++, que possui suporte a orientação a objetos, um importante paradigma de programação e sendo agora tendência atual para o desenvolvimento de firmware. No âmbito da geração de diagramas, pode ser explorada a geração de outros diagramas também úteis a depender do tipo de sistema a ser desenvolvido ou mesmo outras notações visuais, tais como SysML, DFD, DER etc. No âmbito da geração de casos de teste, podem ser exploradas outras técnicas de roteirização e estruturação de casos de teste assim como notações e linguagens específicas para automação, tais como BDD, XUnit e TTCN-3. Além disso, seria possível especificar critérios de teste, tais como análise de valores limites, cobertura do fluxo de controle do programa para abordagens de teste de caixa branca ou cinza etc. Por fim, esses mesmos experimentos podem ser explorados em outras ferramentas de IA Generativa disponíveis ao público, tais como o Gemini do Google, Llama do Facebook e o Claude do Anthropic, para fins de comparação.

Enfim, os resultados obtidos com a aplicação da IA Generativa nas mais diversas áreas tem sido notáveis. Percebe-se um relativo consenso na comunidade, com base nos trabalhos consultados, de que esse tipo de sistema não apresenta ainda as capacidades de superinteligência, mas pode, sim, ser usado para expandir as capacidades humanas em tarefas de criação dos mais diversos de tipos conteúdo. Embora a inteligência artificial amplie as fronteiras do desenvolvimento de software, é na colaboração entre a criatividade humana e o poder da IA que poderá residir o verdadeiro potencial para transformar o futuro da tecnologia.

## REFERÊNCIAS

- AAMANA; AIN, Q. U.; NISA, S. U. Beyond Agile: NLP-Driven Quality Attributes Retrieval Using ChatGPT in Software Development Strategies. 2024. In: **International Conference on Engineering & Computing Technologies (ICECT)**, p. 1-6, 2024.
- BENDER, E. M.; GEBRU; T.; MCMILLAN-MAJOR, A.; SHMITCHELL, S. "On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?". In: **Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency**. New York, NY, USA: Association for Computing Machinery, 2021.
- BROWN, T. et al. Language Models are Few-Shot Learners. In: **Advances in Neural Information Processing Systems**, v. 33, p. 1877–1901, 2020.
- CÁMARA, J.; TROYA, J.; BURGUEÑO, L.; VALLECILLO, A. On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Software and Systems Modeling*, v. 22, n. 3, p. 781–793, 2023.
- CLARK, A.; IGBOKWE, D.; ROSS, S.; ZIBRAN, M. F. A Quantitative Analysis of Quality and Consistency in AI-generated Code. In: **International Conference on Software and System Engineering (ICoSSE)**, 7., 2024, Los Alamitos, CA, USA. IEEE Computer Society, 2024. p. 37-41.
- COMBEMALE, B.; GRAY, J.; RUMPE, B. ChatGPT in software modeling. *Software and Systems Modeling*, v. 22, p. 777-779, 2023.
- COPELAND, Michael. **What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?** NVIDIA Blog, 29 de julho de 2016. Disponível em: <<https://blogs.nvidia.com/blog/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>>. Acesso em: 10 ago. 2024.
- DEVLIN, J.; CHANG, M-W.; LEE, K.; TOUTANOVA, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: **Proceedings of NAACL-HLT**, 2019, p. 4171-4186.
- FENG, Z.; GUO, D.; TANG, D.; DUAN, N.; FENG, X.; GONG, M.; SHOU, L.; QIN, B.; LIU, T.; JIANG, D.; ZHOU, M. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In: COHN, Trevor; HE, Yulan; LIU, Yang (ed.). **Findings of the Association for Computational Linguistics: EMNLP 2020**. Association for Computational Linguistics, 2020. p. 1536-1547. Disponível em: <https://aclanthology.org/2020.findings-emnlp.139>. DOI: 10.18653/v1/2020.findings-emnlp.139. Acesso em: 10 ago. 2024.
- FEIGENBAUM, E. A.; BUCHANAN, B. G.; LEDERBERG, J. On generality and problem solving: A case study using the DENDRAL program. In: **Machine Intelligence**, Vol. 6, p. 165-190, 1971.
- GRAVES, Alex. Supervised Sequence Labelling with Recurrent Neural Networks. **Studies in Computational Intelligence**, v. 385, Springer, 2012.

GUO, M. Java Web Programming with ChatGPT. In: **2024 5th International Conference on Mechatronics Technology and Intelligent Manufacturing (ICMTIM)**. 2024. p. 834-838. DOI: 10.1109/ICMTIM62047.2024.10629560.

HEKLER, Eric. **Understanding the systems we exist in: computing machines as an analogy**. Medium, 5 de agosto de 2020. Disponível em: <https://erichekler.medium.com/understanding-the-systems-we-exist-in-computing-machines-as-an-analogy-4480788f2674>. Acesso em: 10 ago. 2024.

HUSAIN, H.; WU, H.; GAZIT, T.; ALLAMANIS, M.; BROCKSCHMIDT, M. **CodeSearchNet Challenge: Evaluating the State of Semantic Code Search**. 2020. Disponível em: <<https://arxiv.org/abs/1909.09436>>. Acesso em: 10 ago. 2024.

IMANI, S.; DU, L.; SHRIVASTAVA, H. MathPrompter: Mathematical Reasoning using Large Language Models. In: **Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)**, 2023.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd. ed. Upper Saddle River, NJ: Prentice-Hall, 2009.

LI, Y.; SHI, J.; ZHANG, Z. An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering. **IEEE Access**, v. 12, p. 53074-53087, 2024.

LIU, C.; BAO, X.; ZHANG, H.; ZHANG, N.; HU, H.; ZHANG, X.; YAN, M. Guiding ChatGPT for Better Code Generation: An Empirical Study. **IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2024**. p. 102-113.

LIU, P., YU, X., & WANG, C. Prompt Engineering: An Overview. **Journal of Artificial Intelligence Research**, v. 58, p. 56-70, 2023.

OMG (Object Management Group). **Unified Modeling Language (UML) Version 2.5.1**. Needham, MA, USA: Object Management Group, 2017. Disponível em: <<https://www.omg.org/spec/UML/>>. Acesso em: 26 ago. 2024.

MORATIS, K.; DIAMANTOPOULOS, T.; NASTOS, D-N.; SYMEONIDIS, A. Write me this Code: An Analysis of ChatGPT Quality for Producing Source Code. In: **International Conference on Mining Software Repositories, 21., 2024, Lisboa, Portugal**. Association for Computing Machinery, 2024.

MORIMOTO, C. **Guia Foca Linux**. 8. ed. São Paulo: Novatec, 2017.

RABBI, M. F.; CHAMPA, A. I.; ZIBRAN, M. F.; ISLAM, M. R. AI Writes, We Analyze: The ChatGPT Python Code Saga. In: **International Conference on Mining Software Repositories, 21., 2024, Lisboa, Portugal**. Association for Computing Machinery, 2024. p. 177–181.

RAHMANIAR, W. **ChatGPT for Software Development: Opportunities and Challenges**. IT Professional, v. 26, n. 3, p. 80-86, 2024.

SHORTLIFFE, E. H. **Computer-Based Medical Consultations: MYCIN**. New York: Elsevier, 1976.

STALLINGS, W. **Arquitetura e organização de computadores**. 8. ed. São Paulo. Pearson Prentice Hall, 2013.

TANENBAUM, A. S. **Sistemas operacionais modernos**. 3. ed. São Paulo: Pearson, 2009.

TURING, A. M. "Computing Machinery and Intelligence". Mind, 1950.

VASWANI, A.; SHAZER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, Ł.; POLOSUKHIN, I. Attention Is All You Need. In: **Advances in Neural Information Processing Systems 30 (NeurIPS 2017)**. 2017, p. 6000-6008.

WANG, X. et al. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In: **Proceedings of the 11th International Conference on Learning Representations**, 2023.

WEI, J. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: **Proceedings of the 36th International Conference on Neural Information Processing System**, 2024. p. 24824-24837.

WEI, J.; BOSMA, M.; ZHAO, V. Y.; GUU, K.; YU, A. W.; LESTER, B.; DU, N.; DAI, A. M.; LE, Q. V. Zero-shot learning with large pre-trained language models. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021. p. 133-145.

WU, T.; HE, S.; LIU, J.; SUN, S.; LIU, K.; HAN, Q.; TANG, Y. A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development. **IEEE/CAA Journal of Automatica Sinica**, v. 10, n. 5, p. 1122-1136, 2023. DOI: 10.1109/JAS.2023.123618.

YAN, D.; GAO, Z.; LIU, Z. A Closer Look at Different Difficulty Levels Code Generation Abilities of ChatGPT. In: **IEEE/ACM International Conference on Automated Software Engineering (ASE), 38., 2023**, Los Alamitos, CA, USA. IEEE Computer Society, 2023. p. 1887-1898.

ZHANG, Y.; ZHANG, A.; LI, M. SMOLA, A. Automatic Chain of Thought Prompting in Large Language Models. In: **The 11th International Conference on Learning Representations, 2023**.

## ANEXO A – Interação inicial com o ChatGPT

**Você disse:**

Vamos elaborar uma solução de um dispositivo eletrônico para controle de temperatura e disparo de alarmes. Os requisitos serão dados a seguir.

**O ChatGPT disse:**

Ótimo! Vamos detalhar a solução com base nos requisitos. Quais são os requisitos para o dispositivo de controle de temperatura? Se já tiver alguns em mente, podemos começar a partir daí.

**Você disse:**

Estes são os requisitos:

1. Como um usuário do dispositivo, eu quero definir um [valor limite de temperatura] que deve ser armazenado no dispositivo para a realização do [monitoramento de temperatura].
2. Como um usuário do dispositivo, eu quero poder ligar o dispositivo e este iniciar o [monitoramento da temperatura] realizando a leitura do [valor atual da temperatura] de modo contínuo e sem interrupção até que seja desligado.
3. Como um usuário do dispositivo, eu quero que o dispositivo mantenha a [saída digital de alarme] no nível lógico zero caso o [valor limite de temperatura] não tenha sido excedido.
4. Como um usuário do dispositivo, eu quero que o dispositivo mantenha a [saída digital de alarme] no nível lógico um caso o [valor limite de temperatura] tenha sido excedido.
5. Como um usuário do dispositivo, eu quero também que o dispositivo realize uma sinalização luminosa através da piscagem [LED de alarme] quando o [valor limite da temperatura] exceder o [valor atual da temperatura]. Aguarde que irei informá-lo sobre algumas especificações adicionais a seguir.

**O ChatGPT disse:**

Entendido! Temos os requisitos básicos para o controle de temperatura e alarmes, incluindo o armazenamento de um valor limite de temperatura, a leitura contínua do valor atual da temperatura, controle de uma saída digital de alarme e sinalização por LED. Fico no aguardo das especificações adicionais para que possamos elaborar a solução completa.

**Você disse:**

Estas são as especificações adicionais:

1. O [valor limite de temperatura] deve ser um valor entre 0 e 250 graus Celsius.
2. O [valor atual da temperatura] deve ser lido através da conversão analógico digital utilizando o ADC0 canal 0 definido no pino PA0.
3. O [LED de alarme] deve ser implementado através do pino PA5.
4. A [saída digital de alarme] deve ser implementado no pino PA6.
5. A piscagem do [LED de alarme] deve ocorrer a uma frequência de 4 Hertz.

Aguarde que irei informá-lo sobre algumas orientações e exemplos de como iremos gerar código-fonte para esta solução.

**O ChatGPT disse:**

Perfeito! Com essas especificações, já podemos visualizar boa parte do design do sistema. Ficarei aguardando as orientações e exemplos sobre o código-fonte para implementar essa solução.

**Você disse:**

Estas são as orientações e exemplos sobre o código-fonte:

1. Nomes de variáveis globais de escopo de arquivo devem seguir o padrão PascalCase prefixadas com letra 'm' e precedidas com o operador 'static'.

Exemplo de nomes de variáveis globais de escopo de arquivo:

- float mCurrentTemperature = 0;
- uint32\_t mLedBlinkInterval = 125;

2. Nomes de variáveis locais devem seguir o padrão camelCase. Exemplo de nomes de variáveis locais:

- float currentTemperature = 50;
- uint32\_t counter = 0;

3. Nomes de funções de escopo de arquivo devem seguir o padrão camelCase e precedidas com o operador 'static'. Exemplo de nomes de funções de escopo de arquivo:

- static float readTemperature(void);
- static void checkAlarmCondition(void);

4. Nomes de funções de escopo global devem seguir o padrão PascalCase e precedidas pelo substantivo separada por sublinhado pela ação com a ação seguindo o padrão camelCase. Exemplo de nomes de funções de escopo global:

- void Led\_turnOn(void);
- void Alarm\_trigger(void);

5. Os nomes dos arquivos gerados devem seguir o padrão snake\_case. Exemplos de nomes de arquivos no padrão snake\_case:

- led.h e led.c
- alarm\_monitor.h e alarm\_monitor.c

Depois de entender essas orientações, irei realizar uma solicitação a seguir para geração de código-fonte em uma linguagem específica.

**O ChatGPT disse:**

Ótimo! As orientações e exemplos sobre os padrões de nomenclatura e estrutura de código estão bem claras. Estou pronto para a próxima solicitação de geração de código-fonte na linguagem específica que você escolher.

## ANEXO B – Código-fonte gerado inicialmente pelo ChatGPT

Arquivo: main.h

```
#ifndef MAIN_H
#define MAIN_H

#include "stm32g0xx_hal.h"

void SystemClock_Config(void);
void MX_GPIO_Init(void);
void MX_ADC1_Init(void);

#endif /* MAIN_H */
```

Arquivo: main.c

```
#include "main.h"
#include "led.h"
#include "alarm_monitor.h"

/* Variáveis globais de escopo de arquivo */
static float mCurrentTemperature = 0.0;
static float mTemperatureLimit = 100.0; // Padrão inicial, pode ser
configurado pelo usuário
static uint32_t mLedBlinkInterval = 125; // Intervalo de tempo para 4 Hz (125
ms)
static uint8_t mAlarmTriggered = 0; // Indicador de alarme

/* Funções de escopo de arquivo */
static float readTemperature(void);
static void checkAlarmCondition(void);

int main(void) {
    HAL_Init();
    SystemClock_Config();

    // Inicializações de GPIO e ADC
    MX_GPIO_Init();
    MX_ADC1_Init();

    // Loop principal
    while (1) {
        mCurrentTemperature = readTemperature(); // Leitura da temperatura
        checkAlarmCondition(); // Checa condições de alarme
        HAL_Delay(100); // Pequena espera para
suavizar a leitura
    }
}
```

```

    }

}

/***
 * @brief Função para ler a temperatura do sensor no canal ADC0 (PA0)
 * @return float Valor atual da temperatura
 */
static float readTemperature(void) {
    uint32_t adcValue = 0;
    float temperature = 0.0;

    // Start ADC conversion
    HAL_ADC_Start(&hadc1);
    if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) == HAL_OK) {
        adcValue = HAL_ADC_GetValue(&hadc1); // Lê valor ADC
    }
    HAL_ADC_Stop(&hadc1);

    // Conversão do valor ADC para temperatura (simplificado)
    temperature = ((float)adcValue / 4095.0) * 250.0; // Faixa de 0 a 250 °C

    return temperature;
}

/***
 * @brief Função para verificar se o limite de temperatura foi excedido
 * Aciona o LED e a saída de alarme se necessário.
 */
static void checkAlarmCondition(void) {
    if (mCurrentTemperature > mTemperatureLimit) {
        if (!mAlarmTriggered) {
            mAlarmTriggered = 1;
            Alarm_trigger(); // Aciona a saída de alarme
        }
        Led_blink(mLedBlinkInterval); // Piscagem do LED a 4 Hz
    } else {
        if (mAlarmTriggered) {
            mAlarmTriggered = 0;
            Alarm_reset(); // Reseta a saída de alarme
        }
        Led_turnOff();
    }
}

// Configuração do sistema de clock (omitir detalhes aqui para brevidade)
void SystemClock_Config(void) {
    // Configuração típica de clock para STM32
}

```

```

// Inicialização do GPIO
void MX_GPIO_Init(void) {
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitTypeDef GPIO_InitStruct = {0};

    // Configurações do LED (PA5) e saída de alarme (PA6)
    GPIO_InitStruct.Pin = GPIO_PIN_5 | GPIO_PIN_6;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}

// Inicialização do ADC (PA0 - Canal 0)
void MX_ADC1_Init(void) {
    ADC_ChannelConfTypeDef sConfig = {0};

    __HAL_RCC_ADC_CLK_ENABLE();

    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.NbrOfConversion = 1;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
    HAL_ADC_Init(&hadc1);

    sConfig.Channel = ADC_CHANNEL_0;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);
}

```

Arquivo: led.h

```

#ifndef LED_H
#define LED_H

#include "main.h"

void Led_turnOn(void);
void Led_turnOff(void);
void Led_blink(uint32_t interval);

#endif /* LED_H */

```

Arquivo: led.c

```
#include "led.h"

/***
 * @brief Liga o LED de alarme (PA5).
 */
void Led_turnOn(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
}

/***
 * @brief Desliga o LED de alarme (PA5).
 */
void Led_turnOff(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
}

/***
 * @brief Pisca o LED de alarme (PA5) com o intervalo especificado.
 * @param interval Intervalo em milissegundos.
 */
void Led_blink(uint32_t interval) {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5); // Inverte o estado do pino
    HAL_Delay(interval);
}
```

Arquivo: alarm\_monitor.h

```
#ifndef ALARM_MONITOR_H
#define ALARM_MONITOR_H

#include "main.h"

void Alarm_trigger(void);
void Alarm_reset(void);

#endif /* ALARM_MONITOR_H */
```

Arquivo: alarm\_monitor.c

```
#include "alarm_monitor.h"

/***
 * @brief Ativa a saída de alarme digital (PA6).
 */
void Alarm_trigger(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET); // Nível Lógico alto
```

```
}

/***
 * @brief Reseta a saída de alarme digital (PA6).
 */
void Alarm_reset(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET); // Nível Lógico
    baixo
}
```

## ANEXO C – Código-fonte aprimorado pelo ChatGPT

Arquivo aprimorado: led.c

```
#include "led.h"

static uint32_t mLastBlinkTime = 0; // Armazena o último momento em que o LED
piscou

/***
 * @brief Liga o LED de alarme (PA5).
 */
void Led_turnOn(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
}

/***
 * @brief Desliga o LED de alarme (PA5).
 */
void Led_turnOff(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
}

/***
 * @brief Pisca o LED de alarme (PA5) de acordo com o intervalo especificado.
 * @param interval Intervalo em milissegundos.
 */
void Led_blink(uint32_t interval) {
    uint32_t currentTime = HAL_GetTick(); // Obtém o tempo atual em
milissegundos

    // Verifica se o tempo decorrido desde a última piscada é maior que o
intervalo
    if ((currentTime - mLastBlinkTime) >= interval) {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5); // Inverte o estado do LED
        mLastBlinkTime = currentTime; // Atualiza o tempo da última piscada
    }
}
```

## ANEXO D – Código-fonte corrigido pelo ChatGPT

Arquivo corrigido: main.c

```
#include "main.h"
#include "led.h"
#include "alarm_monitor.h"

/* Variáveis globais de escopo de arquivo */
static float mCurrentTemperature = 0.0;
static float mTemperatureLimit = 100.0; // Padrão inicial, pode ser
configurado pelo usuário
static uint32_t mLedBlinkInterval = 125; // Intervalo de tempo para 4 Hz (125
ms)
static uint8_t mAlarmTriggered = 0; // Indicador de alarme

/* Funções de escopo de arquivo */
static float readTemperature(void);
static void checkAlarmCondition(void);

int main(void) {
    HAL_Init();
    SystemClock_Config();

    // Inicializações de GPIO e ADC
    MX_GPIO_Init();
    MX_ADC1_Init();

    // Loop principal
    while (1) {
        mCurrentTemperature = readTemperature(); // Leitura da temperatura
        checkAlarmCondition(); // Checa condições de alarme

        // Não há delay fixo aqui, garantindo que o tempo de piscagem seja
controlado pela função Led_blink
    }
}

/**
 * @brief Função para ler a temperatura do sensor no canal ADC0 (PA0)
 * @return float Valor atual da temperatura
 */
static float readTemperature(void) {
    uint32_t adcValue = 0;
    float temperature = 0.0;

    // Start ADC conversion
```

```

HAL_ADC_Start(&hadc1);
if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) == HAL_OK) {
    adcValue = HAL_ADC_GetValue(&hadc1); // Lê valor ADC
}
HAL_ADC_Stop(&hadc1);

// Conversão do valor ADC para temperatura (simplificado)
temperature = ((float)adcValue / 4095.0) * 250.0; // Faixa de 0 a 250 °C

return temperature;
}

/***
* @brief Função para verificar se o limite de temperatura foi excedido
* Aciona o LED e a saída de alarme se necessário.
*/
static void checkAlarmCondition(void) {
    if (mCurrentTemperature > mTemperatureLimit) {
        if (!mAlarmTriggered) {
            mAlarmTriggered = 1;
            Alarm_trigger(); // Aciona a saída de alarme
        }
        Led_blink(mLedBlinkInterval); // Piscagem do LED a 4 Hz, controlada
        pela função
    } else {
        if (mAlarmTriggered) {
            mAlarmTriggered = 0;
            Alarm_reset(); // Reseta a saída de alarme
        }
        Led_turnOff();
    }
}

```

Arquivo aprimorado: led.c

```

#include "led.h"

static uint32_t mLastBlinkTime = 0; // Armazena o último momento em que o LED
piscou

/***
* @brief Liga o LED de alarme (PA5).
*/
void Led_turnOn(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
}

/***
* @brief Desliga o LED de alarme (PA5).

```

```
 */
void Led_turnOff(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
}

/**
 * @brief Pisca o LED de alarme (PA5) de acordo com o intervalo especificado.
 * @param interval Intervalo em milissegundos.
 */
void Led_blink(uint32_t interval) {
    uint32_t currentTime = HAL_GetTick(); // Obtém o tempo atual em
                                         // milissegundos

    // Verifica se o tempo decorrido desde a última piscada é maior que o
    // intervalo
    if ((currentTime - mLastBlinkTime) >= interval) {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5); // Inverte o estado do LED
        mLastBlinkTime = currentTime; // Atualiza o tempo da última piscada
    }
}
```

## ANEXO E – Código-fonte refeito pelo ChatGPT

Arquivo refatorado: main.c

```
#include "main.h"
#include "led.h"
#include "alarm_monitor.h"

/* Variáveis globais de escopo de arquivo */
static float mCurrentTemperature = 0.0;
static float mTemperatureLimit = 100.0; // Padrão inicial, pode ser
configurado pelo usuário
static uint32_t mAlarmCheckInterval = 50; // Tempo de espera para verificação
de alarme (em ms)
static uint8_t mAlarmTriggered = 0; // Indicador de alarme

/* Funções de escopo de arquivo */
static float readTemperature(void);
static void checkAlarmCondition(void);

int main(void) {
    HAL_Init();
    SystemClock_Config();

    // Inicializações de GPIO, ADC e TIM
    MX_GPIO_Init();
    MX_ADC1_Init();
    MX_TIM1_Init(); // Inicialização do Timer 1

    // Loop principal
    while (1) {
        mCurrentTemperature = readTemperature(); // Leitura da temperatura
        checkAlarmCondition(); // Checa condições de alarme

        HAL_Delay(mAlarmCheckInterval); // Espera configurável para ciclo de
verificação
    }
}

/**
 * @brief Função para ler a temperatura do sensor no canal ADC0 (PA0)
 * @return float Valor atual da temperatura
 */
static float readTemperature(void) {
    uint32_t adcValue = 0;
    float temperature = 0.0;
```

```

// Start ADC conversion
HAL_ADC_Start(&hadc1);
if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY) == HAL_OK) {
    adcValue = HAL_ADC_GetValue(&hadc1); // Lê valor ADC
}
HAL_ADC_Stop(&hadc1);

// Conversão do valor ADC para temperatura (simplificado)
temperature = ((float)adcValue / 4095.0) * 250.0; // Faixa de 0 a 250 °C

return temperature;
}

/***
 * @brief Função para verificar se o limite de temperatura foi excedido
 * Aciona o LED e a saída de alarme se necessário.
 */
static void checkAlarmCondition(void) {
    if (mCurrentTemperature > mTemperatureLimit) {
        if (!mAlarmTriggered) {
            mAlarmTriggered = 1;
            Alarm_trigger(); // Aciona a saída de alarme
            HAL_TIM_Base_Start_IT(&htim1); // Inicia o timer para controlar o
LED
        }
    } else {
        if (mAlarmTriggered) {
            mAlarmTriggered = 0;
            Alarm_reset(); // Reseta a saída de alarme
            HAL_TIM_Base_Stop_IT(&htim1); // Para o timer de controle do LED
            Led_turnOff(); // Certifica que o LED está desligado
        }
    }
}

```

Arquivo refatorado: led.h

```

#ifndef LED_H
#define LED_H

#include "main.h"

void Led_turnOn(void);
void Led_turnOff(void);
void Led_toggle(void);

#endif /* LED_H */

```

Arquivo refatorado: led.c

```
#include "led.h"

/**
 * @brief Liga o LED de alarme (PA5).
 */
void Led_turnOn(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
}

/**
 * @brief Desliga o LED de alarme (PA5).
 */
void Led_turnOff(void) {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
}

/**
 * @brief Alterna o estado do LED de alarme (PA5).
 */
void Led_toggle(void) {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
}
```

Arquivo refatorado: tim.h

```
#ifndef TIM_H
#define TIM_H

#include "main.h"

extern TIM_HandleTypeDef htim1;

void MX_TIM1_Init(void);

#endif /* TIM_H */
```

Arquivo refatorado: tim.c

```
#include "tim.h"

TIM_HandleTypeDef htim1;

/**
 * @brief Inicialização do Timer 1 para controle do LED de alarme.
 */
```

```

void MX_TIM1_Init(void) {
    __HAL_RCC_TIM1_CLK_ENABLE();

    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 64000 - 1; // Prescaler para 1 ms por tick (64
MHz/64000 = 1 kHz)
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 125 - 1; // Período para 125 ms (4 Hz)
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    HAL_TIM_Base_Init(&htim1);

    // Ativar interrupção do TIM1
    HAL_NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
}

/**
 * @brief Interrupção de Timer 1. Piscagem do LED ocorre aqui.
 */
void TIM1_BRK_UP_TRG_COM_IRQHandler(void) {
    HAL_TIM_IRQHandler(&htim1);
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM1) {
        Led_toggle(); // Alterna o estado do LED quando o timer atinge o
período
    }
}

```

## ANEXO F – Código-fonte gerado pelo ChatGPT (especificação atualizada)

Arquivo refatorado: main.c

```
#include "stm32g0xx_hal.h"
#include "temperature_sensor.h"
#include "led.h"
#include "alarm_monitor.h"

// Definição do valor limite de temperatura
#define TEMPERATURE_LIMIT 100.0f

static void SystemClock_Config(void);

int main(void)
{
    HAL_Init();
    SystemClock_Config();

    // Inicializa o LED e o sistema de alarme
    Led_init();
    Alarm_init();
    TemperatureSensor_init();

    while (1)
    {
        // Lê a temperatura atual
        float currentTemperature = TemperatureSensor_read();

        // Verifica a condição de alarme
        Alarm_checkCondition(currentTemperature, TEMPERATURE_LIMIT);

        // Aguarda um pequeno intervalo de tempo
        HAL_Delay(50); // Espera de 50ms
    }
}

// Configuração do clock do sistema
static void SystemClock_Config(void)
{
    // Implementar conforme as especificações do clock desejado
}
```

Arquivo refatorado: alarm\_monitor.h

```
#ifndef ALARM_MONITOR_H
#define ALARM_MONITOR_H

void Alarm_init(void);
void Alarm_checkCondition(float currentTemperature, float temperatureLimit);

#endif // ALARM_MONITOR_H
```

Arquivo refatorado: alarm\_monitor.c

```
#include "alarm_monitor.h"
#include "led.h"
#include "stm32g0xx_hal.h"

// Pino para a saída digital de alarme
#define ALARM_PIN GPIO_PIN_6
#define ALARM_GPIO_PORT GPIOA

static void alarmTrigger(void)
{
    HAL_GPIO_WritePin(ALARM_GPIO_PORT, ALARM_PIN, GPIO_PIN_SET);
    Led_startBlink();
}

static void alarmReset(void)
{
    HAL_GPIO_WritePin(ALARM_GPIO_PORT, ALARM_PIN, GPIO_PIN_RESET);
    Led_stopBlink();
}

void Alarm_init(void)
{
    // Configuração do pino de saída do alarme
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = ALARM_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(ALARM_GPIO_PORT, &GPIO_InitStruct);

    // Inicialmente, o alarme está desligado
    alarmReset();
}

void Alarm_checkCondition(float currentTemperature, float temperatureLimit)
{
    if (currentTemperature > temperatureLimit)
```

```

    {
        alarmTrigger();
    }
    else
    {
        alarmReset();
    }
}

```

Arquivo refatorado: led.h

```

#ifndef LED_H
#define LED_H

void Led_init(void);
void Led_startBlink(void);
void Led_stopBlink(void);

#endif // LED_H

```

Arquivo refatorado: led.c

```

#include "led.h"
#include "stm32g0xx_hal.h"

// Pino para o LED de alarme
#define LED_PIN GPIO_PIN_5
#define LED_GPIO_PORT GPIOA
#define LED_BLINK_FREQUENCY 250 // Período de piscagem de 250 ms (4 Hz)

// Timer utilizado para controle do piscar
TIM_HandleTypeDef htim1;

static void ledBlinkConfig(void)
{
    // Configuração básica do Timer1 para controlar a piscagem
    __HAL_RCC_TIM1_CLK_ENABLE();

    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 8000 - 1;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = LED_BLINK_FREQUENCY - 1;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    HAL_TIM_Base_Init(&htim1);
}

void Led_init(void)
{
    __HAL_RCC_GPIOA_CLK_ENABLE();
}

```

```

GPIO_InitTypeDef GPIO_InitStruct = {0};
GPIO_InitStruct.Pin = LED_PIN;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(LED_GPIO_PORT, &GPIO_InitStruct);

ledBlinkConfig();
}

void Led_startBlink(void)
{
    HAL_TIM_Base_Start_IT(&htim1);
}

void Led_stopBlink(void)
{
    HAL_TIM_Base_Stop_IT(&htim1);
    HAL_GPIO_WritePin(LED_GPIO_PORT, LED_PIN, GPIO_PIN_RESET);
}

// Callback do Timer para alternar o estado do LED
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM1)
    {
        HAL_GPIO_TogglePin(LED_GPIO_PORT, LED_PIN);
    }
}

```

Arquivo refatorado: temperature\_sensor.h

```

#ifndef TEMPERATURE_SENSOR_H
#define TEMPERATURE_SENSOR_H

void TemperatureSensor_init(void);
float TemperatureSensor_read(void);

#endif // TEMPERATURE_SENSOR_H

```

Arquivo refatorado: temperature\_sensor.c

```

#include "temperature_sensor.h"
#include "stm32g0xx_hal.h"

// Canal ADC para Leitura da temperatura
ADC_HandleTypeDef hadc1;

```

```

void TemperatureSensor_init(void)
{
    __HAL_RCC_ADC1_CLK_ENABLE();

    ADC_ChannelConfTypeDef sConfig = {0};

    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
    hadc1.Init.ContinuousConvMode = ENABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
    HAL_ADC_Init(&hadc1);

    // Configuração do canal 0
    sConfig.Channel = ADC_CHANNEL_0;
    sConfig.Rank = ADC_REGULAR_RANK_1;
    sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);

    HAL_ADC_Start(&hadc1);
}

float TemperatureSensor_read(void)
{
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    uint32_t adcValue = HAL_ADC_GetValue(&hadc1);

    // Converter o valor ADC para a temperatura (exemplo: escala de 0-250
    // graus)
    float temperature = (adcValue / 4096.0f) * 250.0f;

    return temperature;
}

```