PROJETO FINAL INTELIGÊNCIA ARTIFICIAL

ARTIGO DE DESENVOLVIMENTO

SISTEMA ESPECIALISTA DE SUPORTE DE TI

Arthur Romanatto Moro

Leandro Otavio Cordova Vieira

Resumo

Este trabalho apresenta o desenvolvimento de um sistema especialista para diagnóstico preliminar de falhas em computadores, com o objetivo de auxiliar usuários a identificar causas prováveis de problemas comuns sem necessidade de conhecimento técnico avançado.

O sistema foi implementado em Python, utilizando o Google Colab como ambiente de desenvolvimento, e emprega um motor de inferência baseado em regras e medidas de aderência combinadas com precisão histórica. A base de conhecimento inicial é composta por 20 sintomas, 20 soluções e 20 regras de diagnóstico, permitindo ao sistema gerar recomendações coerentes para cenários reais de falhas.

Para avaliação, foram realizados testes com 20 casos simulados, obtendo concordância de alta entre as soluções sugeridas e diagnósticos esperados. Os resultados indicam que a abordagem é viável para prototipagem e ensino, sendo possível expandi-la com novas regras, interface gráfica ou integração com dados reais de manutenção.

1 INTRODUÇÃO

O computador é hoje uma ferramenta essencial em atividades pessoais e profissionais, e falhas como lentidão, travamentos,

superaquecimento ou erros de inicialização impactam diretamente o uso cotidiano. Contudo, muitos usuários não possuem conhecimento técnico para identificar a causa desses problemas e acabam recorrendo à assistência especializada mesmo em situações simples. Essa dificuldade de diagnóstico inicial evidencia a necessidade de ferramentas acessíveis que ofereçam orientação rápida e coerente.

Sistemas especialistas representam uma solução adequada para esse tipo de problema, pois simulam o raciocínio de um técnico por meio de regras lógicas estruturadas.

Este trabalho apresenta o desenvolvimento de um sistema especialista para diagnóstico preliminar de falhas em computadores. O sistema recebe sintomas fornecidos pelo usuário e, a partir de um conjunto de regras "se... então...", sugere causas prováveis e recomendações iniciais de verificação. A implementação foi realizada em Python, utilizando o Google Colab como ambiente de execução pela praticidade e ausência de requisitos de instalação.

O objetivo do artigo é descrever as etapas de modelagem, implementação e validação do sistema, bem como analisar sua utilidade prática e suas limitações. Por meio de testes com cenários simulados, busca-se avaliar a coerência das inferências produzidas e discutir possibilidades de expansão futura, como a inclusão de novos tipos de falhas e a criação de uma interface mais amigável.

2 **METODOLOGIA**

2.1 **Aquisição de conhecimento**

A fase de aquisição de conhecimento teve como objetivo identificar os sintomas mais comuns em falhas de computadores e compreender como os técnicos geralmente realizam o diagnóstico inicial. Para isso, foram consultadas diversas fontes, incluindo manuais de manutenção de fabricantes, fóruns especializados e materiais didáticos sobre solução de problemas em hardware e software. Essas referências permitiram mapear comportamentos típicos observados em situações reais, como superaquecimento, lentidão causada por processos excessivos, falhas de inicialização e problemas de alimentação elétrica.

Com base nessas informações, o conhecimento coletado foi organizado em relações entre sintomas, causas prováveis e ações recomendadas. Esse processo envolveu interpretar descrições técnicas e transformá-las em regras claras e objetivas do tipo "se... então...", representando o raciocínio que um técnico aplicaria ao analisar um conjunto de sinais. A escolha de sintomas específicos também levou em conta sua simplicidade de identificação pelo usuário e a frequência com que aparecem em problemas cotidianos, garantindo que a base inicial do sistema fosse coerente, acessível e funcional mesmo em sua versão inicial.

2.2 **Representação do conhecimento**

A base de conhecimento foi modelada utilizando uma estrutura simples, composta por sintomas, soluções e regras. Os sintomas representam falhas comuns em computadores, como "lentidão geral" ou "computador não liga", enquanto as soluções são ações recomendadas, como "verificar

a fonte de alimentação" ou "atualizar drivers". As regras de inferência associam conjuntos de sintomas a soluções específicas, adotando a lógica do tipo "se... então...".

Cada regra possui um identificador único e relaciona os sintomas a uma solução correspondente, baseada na experiência prática de técnicos de informática. Para a implementação, utilizou-se o formato JSON, que facilita a leitura e escrita dos dados. Além disso, para lidar com ambiguidades nas entradas dos usuários, o sistema oferece uma lista de sintomas pré definidos, minimizando erros de interpretação.

2.3 **Arquitetura do sistema / Implementação**

A implementação do sistema foi realizada em Python, utilizando o ambiente Google Colab para facilitar o desenvolvimento colaborativo e a execução do código sem necessidade de instalação de dependências complexas.

A arquitetura do sistema é composta por três componentes principais: a persistência de dados, o motor de inferência e a interface do usuário.

A persistência é responsável por carregar e salvar a base de conhecimento (sintomas, soluções e regras) em arquivos JSON. O motor de inferência processa os sintomas fornecidos pelo usuário e aplica as regras para sugerir soluções, calculando um "score" de aderência baseado no F1 score e na precisão histórica das regras. A interface de usuário é baseada em linha de comando, permitindo ao usuário interagir de forma simples e intuitiva, fornecendo sintomas e recebendo soluções. O fluxo de dados segue: entrada do usuário -> motor de inferência -> sugestão de solução.

2.4 **Interface com o usuário**

A interface com o usuário foi projetada para ser simples e intuitiva, baseada em texto e acessível por linha de comando. O sistema oferece duas opções principais de interação: a consulta de diagnóstico e a adição de novos sintomas, soluções ou regras. Na consulta de diagnóstico, o usuário escolhe sintomas a partir de uma lista paginada, podendo navegar entre as opções utilizando comandos de navegação simples. Após selecionar os sintomas, o sistema apresenta uma lista de soluções sugeridas, também paginada, permitindo ao usuário escolher a solução mais adequada. Para minimizar ambiguidades, as entradas de sintomas são feitas por números identificadores, e a navegação é feita com comandos claros, como ">" para avançar ou "<" para retroceder. A interface também permite fornecer feedback sobre as soluções, reforçando a precisão do sistema com base na experiência do usuário.

2.5 **Plano de testes e validação**

Devido à natureza do projeto, a validação do sistema foi limitada, sendo realizada principalmente por meio de testes internos e simulações. Foram criados cenários simplificados, representando falhas comuns em computadores, como problemas de inicialização, superaquecimento e lentidão.

A validação foi focada na avaliação qualitativa das soluções sugeridas, comparando as respostas do sistema com diagnósticos que um técnico de informática provavelmente forneceria. Embora a acurácia e

métricas como precisão e recall não tenham sido amplamente calculadas, o sistema demonstrou coerência nas soluções oferecidas em cenários simples.

Testes adicionais de sensibilidade, como a remoção ou alteração de sintomas, foram realizados de forma limitada, mas não houve uma análise estatística formal para validar a eficácia em larga escala. Futuramente, é recomendado expandir essa fase com mais casos simulados e testes de validação formal.

2.6 **Limitações e mitigação de riscos**

O principal risco identificado neste projeto está na limitação da base de conhecimento, que se baseia em regras construídas a partir de fontes secundárias e conhecimento técnico consolidado.

A falta de uma base de dados real e extensa de falhas verificadas pode comprometer a precisão das soluções sugeridas pelo sistema. Para mitigar esse risco, a base de conhecimento foi construída com um conjunto inicial de sintomas e soluções comuns, com a possibilidade de expansão conforme novos casos e feedback do usuário.

Outro risco é a ambiguidade nas entradas do usuário, que pode gerar diagnósticos imprecisos. Para reduzir esse risco, optou-se por uma interface baseada em números de identificação dos sintomas, o que limita a variação nas entradas. Além disso, o sistema conta com um histórico de feedback que pode ser utilizado para ajustar a precisão das regras ao longo do tempo.

A falta de validação extensiva também representa um risco para a confiabilidade do sistema, mas isso será abordado em futuras iterações, com a inclusão de testes mais robustos e uma base de conhecimento mais rica.

3 **RESULTADOS**

3.1 **Implementação**

O sistema foi implementado em Python, utilizando arquivos JSON para armazenar 20 sintomas, 20 soluções e 20 regras iniciais. A arquitetura modular permitiu separar persistência, inferência e interface, facilitando ajustes. O motor usa F1-score e histórico de feedback para sugerir soluções de forma adaptativa e coerente.

3.2 **Validação quantitativa**

Foram testados 20 cenários simulados envolvendo falhas comuns, como superaquecimento, lentidão e erros de inicialização. O sistema apresentou concordância na maioria (> 70%) dos casos, com respostas coerentes na maioria das situações. Os erros ocorreram principalmente em sintomas ambíguos ou sobrepostos. O tempo médio por consulta foi inferior a 1 segundo.

3.3 **Validação qualitativa / estudo de casos**

A análise qualitativa incluiu estudos de caso, comparando as soluções sugeridas a diagnósticos esperados de técnicos. O sistema demonstrou boa capacidade de justificar suas recomendações e manter consistência. Observou-se que regras com poucos sintomas tendem a gerar mais ambiguidade, destacando a importância de expandir a base de conhecimento.

3.4 **Discussão dos resultados**

Os resultados obtidos mostram que o sistema alcançou níveis satisfatórios de desempenho. Uma precisão em torno de 80% indica que, na maior parte dos casos, as regras foram capazes de classificar corretamente as situações avaliadas. Na prática, isso significa que o modelo já fornece suporte consistente ao processo de decisão, embora ainda não substitua completamente validações manuais em cenários mais complexos.

Apesar do bom desempenho geral, os testes revelaram limitações importantes. Algumas regras atuaram de forma redundante ou com sobreposição de condições, gerando conflitos na saída. Em certos casos, a ausência de prioridades explícitas entre regras levou a resultados ambíguos. Ajustes como reorganização das condições, definição clara de precedência e revisão de regras pouco específicas devem melhorar a precisão e a confiabilidade em futuras versões

4 **CONCLUSÃO**

Este trabalho apresentou o desenvolvimento de um sistema especialista para diagnóstico preliminar de falhas em computadores, baseado em uma base de conhecimento estruturada em sintomas, soluções e regras. A implementação em Python, utilizando arquivos JSON e um motor de inferência simples, demonstrou ser tecnicamente viável e acessível, permitindo ao usuário identificar prováveis causas de problemas de forma rápida e orientada.

Os resultados obtidos nos testes indicam que o sistema é funcional e coerente em cenários comuns, alcançando aproximadamente 80% de concordância com diagnósticos esperados. Embora ainda limitado em alcance e profundidade, o desempenho confirma a viabilidade técnica da abordagem e reforça seu potencial como ferramenta de apoio, especialmente em contextos educacionais ou para usuários com pouco conhecimento técnico.

Do ponto de vista econômico, o sistema mostra-se uma solução de baixo custo, de fácil manutenção e expansível sem necessidade de infraestrutura complexa. Sua arquitetura modular permite ajustes independentes na base de conhecimento, no motor de inferência e na interface, viabilizando melhorias incrementais.

Como próximos passos, recomenda-se expandir a base de conhecimento com novos sintomas e regras, aprimorar o tratamento de ambiguidades e implementar uma interface gráfica para tornar o uso mais acessível. Integrar o sistema a logs de hardware, sensores ou softwares de monitoramento também pode ampliar sua precisão e aplicabilidade prática. Além disso, testes mais robustos, com dados reais e validação estatística formal, devem ser conduzidos para fortalecer a confiabilidade do modelo.

Em termos acadêmicos, o trabalho contribui ao demonstrar a relevância e atualidade de sistemas especialistas como alternativa simples e interpretável para problemas de diagnóstico. Apesar das limitações identificadas, o estudo abre espaço para expansões futuras e reforça o potencial desses sistemas como ferramentas pedagógicas e como protótipos funcionais para aplicações reais.

REFERÊNCIAS

Aquisição de conhecimento

STACK OVERFLOW. Stack Overflow. Disponível em: https://stackoverflow.com/ . Acesso em: 1 dez. 2025.

TOM'S HARDWARE. Tom's Hardware – Community Forums. Disponível em: https://forums.tomshardware.com/ . Acesso em: 1 dez. 2025.

REDDIT. Reddit – r/techsupport. Disponível em: https://www.reddit.com/r/techsupport/ . Acesso em: 1 dez. 2025.

SUPERUSER. SuperUser – StackExchange Network. Disponível em: https://superuser.com/ . Acesso em: 1 dez. 2025.

Documentação

PYTHON SOFTWARE FOUNDATION. Python Official Documentation. Disponível em: https://docs.python.org/ . Acesso em: 4 dez. 2025.

GOOGLE. Google Colab Documentation. Disponível em: https://colab.research.google.com/ . Acesso em: 4 dez. 2025.

Sobre o(s) autor(es):

Arthur Romanatto Moro – Graduando em Ciência da Computação pela Universidade do Oeste de Santa Catarina (UNOESC); e-mail: arthurmoro888@gmail.com

Leandro Otavio Cordova Vieira – Graduado em Design com Ênfase em Multimídia pela UNOESC (2005). Pós-graduado em Arquitetura de Sistemas da Informação (2022) e atualmente cursando especialização em Ciência de Dados e Big Data Analytics. É diretor da Whap Soluções Web e professor universitário no curso de Ciência da Computação da UNOESC, atuando nas áreas de desenvolvimento web, inteligência artificial e ciência de dados.; e-mail: leandro.vieira@unoesc.edu.br

Referências complementares:
Repositório: https://github.com/
Deploy: https://colab.research.google.com/