



Diplomová práce

Využití AI v softwarovém inženýrství

Studijní program:

N0714A270010 Mechatronika

Autor práce:

Bc. Daniel Dominko

Vedoucí práce:

Ing. Roman Špánek, Ph.D.

Ústav mechatroniky a technické informatiky

Liberec 2024



Zadání diplomové práce

Využití AI v softwarovém inženýrství

<i>Jméno a příjmení:</i>	Bc. Daniel Dominko
<i>Osobní číslo:</i>	M22000042
<i>Studijní program:</i>	N0714A270010 Mechatronika
<i>Zadávací katedra:</i>	Ústav mechatroniky a technické informatiky
<i>Akademický rok:</i>	2023/2024

Zásady pro vypracování:

1. Seznamte se s tématy velkých jazykových modelů, transformerů, neuronových sítí a softwarového inženýrství.
2. Nastudujte současný stav vývoje vybraných nástrojů souhrnně označovaných jako AI, jako jsou například:
ChatGPT, Google Bard, Microsoft Azure chatbot, HuggingChat, GitHub Copilot a další.
3. Připravte detailní porovnání výhod a nevýhod takovýchto technologií při vývoji softwarového díla na jednotlivých krocích vývoje, konkrétně pak analýzy, návrhu, implementace, testování a CI/CD.
4. Porovnejte vývoj softwarového díla s použitím AI a bez jejich využití, vyzdvihněte výhody a nevýhody těchto technologií.

Rozsah grafických prací: dle potřeby dokumentace
Rozsah pracovní zprávy: 40 až 50 stran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: čeština

Seznam odborné literatury:

- [1] PAVLÍČEK, Antonín a SYROVÁTKOVÁ, Jana. Základy moderní informatiky. [Průhonice]: Professional Publishing, 2022. ISBN 978-80-88260-59-2.
- [2] SHNEIDERMAN, Ben. Human-centered AI. Oxford: Oxford University Press, [2022]. ISBN 978-0-19-284529-0.
- [3] PAN, Chao. Deep learning fundamentals: an introduction for beginners. [Wilmington, Delaware]: AI Sciences, 2018. ISBN 978-1-7212-3088-4.

Vedoucí práce: Ing. Roman Špánek, Ph.D.
Ústav mechatroniky a technické informatiky

Datum zadání práce: 12. října 2023
Předpokládaný termín odevzdání: 14. května 2024

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

doc. Dr. Ing. Jaroslav Hlava
garant studijního programu

V Liberci dne 12. října 2023

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

Poděkování

Rád bych v první řadě vyjádřil své díky Ing. Romanu Špánkovi, Ph.D., za vypsání takto fascinujícího a netradičního tématu, a za ověření prvních prototypů aplikace i samotné práce. Dále bych chtěl vyjádřit vděk společnosti OpenAI za poskytnutí svého modelu ChatGPT volně k dispozici, čímž bylo možné realizovat tuto práci. Totéž platí i o platformě Huggingchat, která mi poskytla zázemí pro testování open-source modelů, jež bych nemohl spustit lokálně. Nakonec bych chtěl poděkovat mnoha YouTube kanálům, které mi pomohly porozumět fungování velkých jazykových modelů do velmi detailní úrovně, a také své rodině a přátelům za ověření práce a podporu během jejího vytváření.

Využití AI v softwarovém inženýrství

Abstrakt

Práce se zaměří na využití AI v prostředí softwarového inženýrství. Nejdříve uvede problematiku neuronových sítí, kódování slov do vektorů a architektury transformerů. Dále se zaměří na nejlepší techniky práce s AI. Poté porovná použití této technologie na vývoji webové aplikace, a to ve fázích analýzy, návrhu, implementace, testování a CI/CD. Nakonec shrne výhody a nevýhody používání takových technologií při vývoji software.

Klíčová slova: Umělá inteligence, AI, Softwarové inženýrství, Neuronové sítě, Kódování slov do vektorů, Transformeři

Usage of AI in software engineering

Abstract

This thesis will focus on using AI in software engineering. First, it will focus on neural networks, word embeddings, and transformer models. Next, it will focus on best practices for prompting large language models. Then, it will evaluate the usage of this technology throughout the software development lifecycle on a test web app, mainly in analysis, design, implementation, testing, and CI/CD. In the end, we will evaluate the benefits of using said technologies in software development.

Keywords: Artificial intelligence, AI, Neural networks, Word embeddings, Software engineering, Transformers

Obsah

Seznam zkratk	8
Seznam obrázků	9
1 Úvod	10
2 Neuronové sítě	11
2.1 Základní pojmy	11
2.1.1 Aktivační funkce	11
2.1.2 Vícevrstvé perceptrony (MLP)	12
2.2 Učení neuronové sítě	12
2.2.1 Chybová funkce	13
2.2.2 Optimizéry	13
2.2.3 Zpětná propagace	14
2.2.4 Trénování neuronové sítě	14
3 Zpracování jazyka	16
3.1 Parametry modelů	16
3.2 Tokenizace	17
3.3 Embedding modely	17
3.4 Transformer modely	19
3.4.1 Kódování pozice	19
3.4.2 Masked self-attention	20
3.5 Techniky použití AI	21
3.5.1 Prompt engineering	21
3.5.2 Hledání v dokumentech	22
4 Vytvoření projektu s AI	23
4.1 Úvodem...	23
4.2 Hodnocení	23
4.3 Testovací aplikace	24
4.4 Softwarové inženýrství	25
4.5 Analýza problému	25
4.6 Analýza s pomocí AI	26
4.6.1 Otázky na zákazníka	26
4.6.2 Upřesnění aplikace	28
4.6.3 Architektura projektu	30

4.6.4	Přídavné prvky	31
4.6.5	Analýza rizik	32
4.6.6	Hodnocení dle seniority	33
4.7	Návrh struktury aplikace	33
4.8	Návrh s pomocí AI	34
4.8.1	Doporučené technologie	34
4.8.2	Specifikace návrhu	36
4.8.3	UML	39
4.8.4	Hodnocení dle seniority	40
4.9	Implementace s pomocí AI	40
4.9.1	Agenti	44
4.9.2	Hodnocení dle seniority	46
4.10	Zlepšování kódu s pomocí AI	46
4.10.1	Zlepšování implementace	47
4.10.2	Zlepšování testů	52
4.10.3	Hodnocení dle seniority	55
4.11	Testování	55
4.12	Testování s pomocí AI	55
4.12.1	Hodnocení dle seniority	58
4.13	Alternativní kopiloti	58
4.13.1	CodiumAI	58
4.13.2	Cody	59
4.13.3	Cursor	59
4.14	Nasazení do produkce s pomocí AI	59
4.14.1	Hodnocení dle seniority	60
5	Závěr	62
	Použitá literatura	73

Seznam zkratk

SDLC – Software Development Life Cycle, životní cyklus vývoje software
AI – Artificial Intelligence, umělá inteligence
CI/CD – Continuous Integration / Continuous Development
API – Application Programming Interface, programové rozhraní aplikace
IDE – Integrated Development Environment, vývojové prostředí
JSON – JavaScript Object Notation
HTML – Hyper Text Markup Language
CSRF – Cross Site Request Forgery
UX – User Experience, uživatelský zážitek
UML – Unified Modeling Language, unifikovaný modelovací jazyk
RAG – Retrieval Augmented Generation, hledání v dokumentech
EOS – End Of Sequence, konec sekvence
NLP – Natural Language Processing, počítačové zpracování jazyka
MNIST – Modified National Institute of Standards and Technology database
LLM – Large Language Model, velký jazykový model
LRU – Least Recently Used, posledně použito
PaLM – Pathways Language Model
GPT – Generative pre-trained transformer
LLaMA – Large Language Model Meta AI
CoT – Chain of Thoughts, řetězec myšlenek
ToT – Tree of Thoughts, strom myšlenek
BPE – Byte Pair Encoding, zakódování bajtových dvojic
WCAG – Web Content Accessibility Guidelines, příručka přístupnosti na webu
PEP – Python Enhancement Proposals, návrhy zlepšení Pythonu
ECMAScript – European Computer Manufacturers Association Script
YAML – YAML Ain't Markup Language

Seznam obrázků

2.1	Blokové schéma neuronu [2]	11
2.2	Jednoduchá neuronová síť pro rozpoznání číslic [6]	12
3.1	Ukázka embedding vektorů [20]	18
3.2	Model pro vytvoření embedding vektorů [21]	19
3.3	Blokové schéma transformeru [28]	21
4.1	UML diagram aplikace od Claude [1]	39

1 Úvod

Tato diplomová práce se zaměřuje na analýzu výhod a nevýhod využití umělé inteligence (AI) při návrhu webové aplikace. Cílem této práce je provést rozbor současného stavu dostupných nástrojů umělé inteligence v oblasti softwarového inženýrství. Hodnocení proběhne v jednotlivých etapách vývoje aplikace, tj. analýze, návrhu, implementaci, testování a kontinuální integraci a doručování (CI/CD).

Pro lepší pochopení problematiky nejdříve podrobněji prozkoumáme fungování neuronových sítí. Jak funguje neuronová síť v základní podobě? Jak se učí neuronová síť předpovídat počasí, rozpoznávat osoby na obrázcích nebo generovat kód? Jaké matematické operace jsou klíčové při trénování neuronových sítí?

Dále se zaměříme na neuronové sítě specializované na zpracování lidského jazyka a na velké jazykové modely (LLM), jako je například ChatGPT. Prozkoumáme postup, jak převedeme slova do podoby, které neuronová síť dokáže porozumět, pomocí zakódování významu do vektorů a tokenizace. Následně si vysvětlíme architekturu transformerů, která je základem většiny velkých jazykových modelů. Nakonec se zaměříme na techniky optimalizace vstupních dat, abychom dosáhli co nejlepších výsledků z modelu, včetně metody „prompt engineering“ a pokročilých přístupů, jako je doplňování kontextu modelu (RAG).

Následuje vytvoření projektu s využitím umělé inteligence a hodnocení jejího přínosu v jednotlivých fázích vývoje softwaru. Bude specifikován rozsah testované aplikace, kritéria a metodika hodnocení. Hodnocení bude zohledňovat přínos umělé inteligence z hlediska juniorní a mediorní úrovně vývoje. Dále budeme hodnotit i obecnější hlediska, jako je například aktuálnost a relevance odpovědí k dané otázce. V závěru zhodnotíme přínos neuronových sítí při návrhu aplikace v jednotlivých fázích vývoje softwaru, a také poskytneme pohled do budoucnosti využití AI jako pomocníka programátora, případně jeho náhradu.

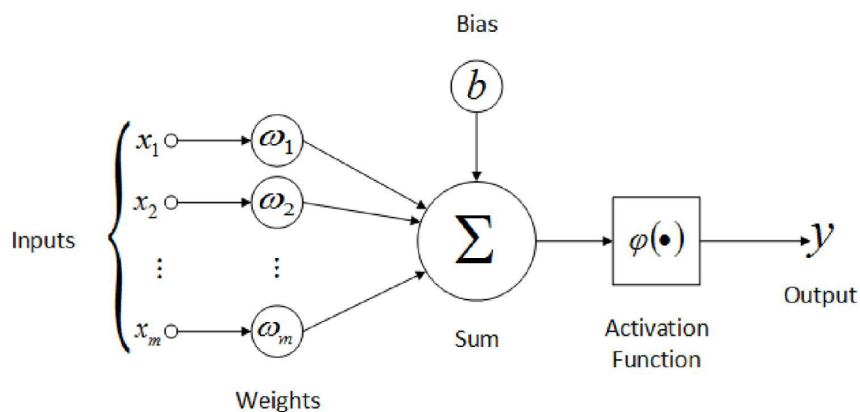
2 Neuronové sítě

2.1 Základní pojmy

Neuronová síť se skládá z neuronů. Propojení mezi neurony mají určenou váhu a doplněk (bias). Tato struktura čerpá inspiraci z lidského mozku, který obsahuje miliardy propojených biologických neuronů [1]. Neuron se skládá z vstupů x_i , aktivační funkce a výstupu. Vstupy x_i jsou násobeny vahou w_i , která určuje důležitost vstupu. Celkový výstup lze dále upravit doplňkem b . Výstup neuronu je definován následující rovnicí:

$$\sum_{i=1}^n w_i x_i + b \quad (2.1)$$

Výstup je následně odeslán do aktivační funkce $\phi(v)$. Tato funkce může, ale nemusí, upravit hodnotu výstupu. Výstup může být dále předán buď dalšímu neuronu nebo přímo na výstup celé neuronové sítě. Tento proces se provádí napříč všemi neurony v síti a nazývá se dopředná propagace [1].



Obrázek 2.1: Blokové schéma neuronu [2]

2.1.1 Aktivační funkce

Jedná se o matematickou funkci, která upravuje výstup neuronu [3]. Příkladem může být pravděpodobnost určité události. Pokud je výstup neuronu v řádu tisíců, nelze bez aktivační funkce získat z sítě hodnotu mezi 0 a 1. Pro

převedení nám poslouží funkce sigmoidy. Existuje několik typů aktivačních funkcí, například ReLU, která vrací hodnotu pouze v případě, že je kladná, záporné hodnoty jsou nulovány.

$$f(x) = \max(0, x) \quad (2.2)$$

Pokud bychom měli určit pravděpodobnost více než dvou událostí, použili bychom funkci softmax, která převádí vektor hodnot na pravděpodobnostní distribuci [4].

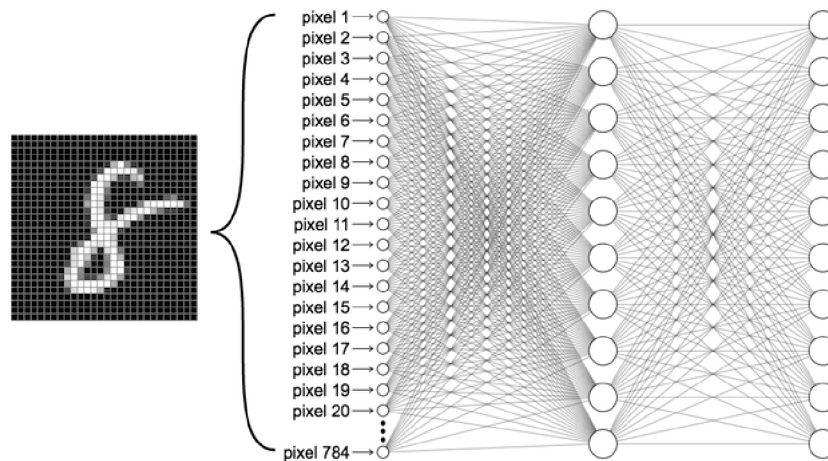
$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad (2.3)$$

2.1.2 Vícevrstvé perceptrony (MLP)

Neurony jsou uspořádány do vrstev. Neuronová síť se skládá z libovolného počtu skrytých vrstev umístěných mezi vstupní a výstupní vrstvou [5]. Jedním z příkladů využití je rozpoznávání číslic na datasetu MNIST.

Vstupní vrstva přijímá hodnoty šedotónových pixelů uvnitř obrázku, které jsou redukovány z matice na vektor. Skryté vrstvy se naučí charakteristiky obrázku, jako jsou například křivky, což umožní rozpoznat, zda se jedná o číslo 1 nebo 8. Výstupní vrstva pak identifikuje danou číslici.

Tento typ sítě je nazýván vícevrstvným perceptronem (MLP) [6]. Velikost výstupní vrstvy může být libovolná. Důležité je, aby výstup z vrstvy měl stejnou dimenzi jako vstup následující vrstvy, a to po celé síti [6].



Obrázek 2.2: Jednoduchá neuronová síť pro rozpoznání číslic [6]

2.2 Učení neuronové sítě

V učení neuronových sítí hrají klíčovou roli data, která obsahují správné odpovědi na to, co se snažíme neuronovou sítí předpovědět. Tento typ učení

je nazýván učením s učitelem. Kvalita dat přímo ovlivňuje kvalitu konečné předpovědi neuronové sítě.

Data musí být pečlivě předzpracována tak, aby byla maximalizována pravděpodobnost správné předpovědi. Po zpracování dat získáváme rysy dat, nejjednodušeji si je lze představit sloupce v tabulárních datech.

Při rozdělení dat musíme ale dávat pozor například na bias, který nastává, pokud data nemají ekvivalentní počet bodů z různých skupin. V kontextu velkých jazykových modelů by to například mohlo znamenat nerovnoměrné rozložení programovacích příspěvků a příspěvků o zahradničení v datech, to ale může být žádoucí, pokud trénujeme model na programování, a ne na zahradničení [7].

Data procházejí různými vrstvami neuronů v procesu dopředné propagace, jak bylo zmíněno výše. Následně se provádí zpětná propagace, kdy probíhá učení neuronové sítě. Jedná se o proces, při kterém se iterativně upravují váhy a doplňky tak, aby síť provedla předpověď co nejpodobnější správným odpovědím [7].

2.2.1 Chybová funkce

Pro pochopení zpětné propagace je ale potřeba vědět, co je to chybová funkce. Jedná se o funkci, která měří rozdíl mezi správnou a předpovězenou hodnotou a „penalizuje“ neuronovou síť za špatné předpovědi. Účelem trénování je minimalizovat hodnotu chybové funkce, což znamená, že síť se snaží najít takové parametry (váhy), které minimalizují rozdíl mezi jejími předpověďmi a skutečností.

Chybovou funkcí může být např. odmocnina střední kvadratické odchylky, definována tímto vzorcem [8].

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (2.4)$$

Kdy n značí počet dat, \hat{y}_i značí předpovězenou hodnotu a y_i značí cílovou hodnotu.

2.2.2 Optimizéry

Optimizéry jsou algoritmy používané k aktualizaci vah neuronové sítě během trénování s cílem minimalizovat chybovou funkci. Nejčastěji používaným optimizérem je metoda největšího spádu (gradient descent), který využívá gradientu chybové funkce k určení směru nejrychlejšího poklesu chyby. Existuje několik variant tohoto algoritmu, které se liší v tom, jakým způsobem aktualizují váhy sítě, jako např. SGD nebo Adam [9].

$$\theta_{j+1} = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta} \quad (2.5)$$

2.2.3 Zpětná propagace

Zpětná propagace je klíčovým procesem v trénování neuronových sítí. Spočívá v aktualizaci vah sítě tak, aby minimalizovaly chybovou funkci. Tento proces využívá derivace chybové funkce podle vah sítě. Každá vrstva sítě obdrží část chyby a vypočte gradienty chybové funkce. Tyto gradienty určují směr a velikost, jakým by se měly váhy aktualizovat, aby se minimalizovala chybová funkce. Zpětná propagace umožňuje sítím učit se a zlepšovat své schopnosti předvídání na základě trénovacích dat [10] [11].

$$\frac{\delta J}{\delta w_n} \quad (2.6)$$

2.2.4 Trénování neuronové sítě

Během trénování neuronové sítě jsou data nejprve předána síti, která provede přepočítání hodnot všech vrstev s aktuálními vahami. Následně je vypočtena chybová funkce, která měří rozdíl mezi předpovězenými a skutečnými výsledky. Poté probíhá zpětná propagace chyby, během které jsou gradienty chybové funkce vypočteny pomocí algoritmu zpětné propagace, což umožňuje aktualizaci vah sítě tak, aby minimalizovaly chybu predikce. Tento proces se opakuje pro vícero dat, dokud není dosaženo dostatečné úrovně přesnosti [12].

Zmíněný proces je ale pouze jedním krokem při trénování jednoho neuronu uvnitř neuronové sítě. Kroků je ale potřeba tisíce, na miliardách neuronů, než se neuronová síť naučí správně předpovídat. Jedná se také o proces „trénování“ neuronové sítě. Jak ale natrénovanou síť použijeme pro předpověď? Neznámá hodnota je vstupem do natrénované neuronové sítě. Následuje proces dopředné propagace a výstupem je předpověď pro danou hodnotu [12].

To je důležité v procesu validace, kdy před procesem trénování odebereme část našich dat, jejichž výstup následně otestujeme na natrénované síti. Tím lze zhodnotit schopnost předpovídat neznámá data. Při procesu validace lze využít další funkce, které určí kvalitu neuronové sítě. Pro pravděpodobnostní modely by se mohlo jednat o přesnost, tedy podíl správných předpovědí ku všem předpovědím neuronové sítě. Obecně je možné využít jakoukoliv metriku, kterou lze vypočítat z výstupů neuronové sítě [12].

Jedna „epocha“ trénování značí bod, kdy síť využila všechna trénovací data. Trénink se může pohybovat od pár epoch po několik stovek. Trénovací data jsou s každou epochou náhodně rozdělena, aby se síť nenaučila předpovídat pouze pořadí dat. Jak moc je potřeba síť trénovat se odvíjí od prvně nastavených náhodných hodnot [12].

Hodnoty vah lze uložit do binárního formátu, čehož lze využít k „přenášenému učení“. Myšlenkou je využití existující neuronové sítě s natrénovanými vahami pro podobný úkol [13]. Síť tak akorát při tréninku „doladíme“ na našich datech a tím zlepšíme kvalitu pro náš úkol [14]. Tato metoda je obzvláště

využívaná u velkých jazykových modelů, jenž jsou velice finančně náročné na trénink, ale ne tak moc na odladění.

3 Zpracování jazyka

Doména zpracování jazyka (NLP) s pomocí strojového učení, případně neuronových sítí je o mnoho starší, než velké jazykové modely jako GPT nebo LLaMA. Pro pochopení LLM potřebujeme znalost „embedding“ a „transformer“ modelů, spolu s procesem „tokenizace“. Při tom si i vysvětlíme proces od předání vstupu LLM, až po výslednou odpověď.

3.1 Parametry modelů

Předtím, než se podíváme na transformery jako takové, je dobré si přiblížit různé parametry, které hrají roli v kvalitě modelu. Velikost modelu označuje, jak počet jeho parametrů, což zvyšuje jeho kvalitu, tak i např. velikost na disku, což zvětšuje obecnou náročnost modelu [15].

Neméně důležitým faktorem jsou data, na kterých byl model trénován. Například v průběhu práce zjistíme, že menší model, jako je CodeLLaMA-34B oproti Falcon-180B, podává lepší výsledky, protože byl, dle dostupných informací, trénován hlavně na programovacích úlohách [15] [16].

Dalším důležitým parametrem je kontextové okno, neboli počet tokenů, které model uvažuje při tvorbě odpovědi. Tento parametr se může pohybovat od několika set po stovky tisíc. V našem testování jsme nedosáhli limitu u žádného modelu, ale pokud bychom například chtěli analyzovat dokumentaci knihovny, kontextové okno o sto tisících tokenech by přišlo vhod [15].

Teplota v případě velkých jazykových modelů určuje míru náhodnosti předpovědi dalšího slova. Vyšší teplota zplošťuje distribuci pravděpodobností z funkce softmax, což zvyšuje šanci na výběr méně pravděpodobného slova [15].

Nebo tzv. „systémový prompt“, který umožňuje předdefinovat formát odpovědi umělé inteligence a omezit ji. Tyto prompty se dávají do speciálního bloku textu, odděleného speciálními tokeny, aby model pochopil, že se jedná o systémový prompt.

Nastavení teploty a systémového promptu je možné pouze u CodeLLaMA a Claude-instant, zatímco ChatGPT umožňuje nastavit pouze systémový prompt, ale u ostatních chatbotů tyto nastavení, v čase vytváření aplikace, nebyly k dispozici [15].

3.2 Tokenizace

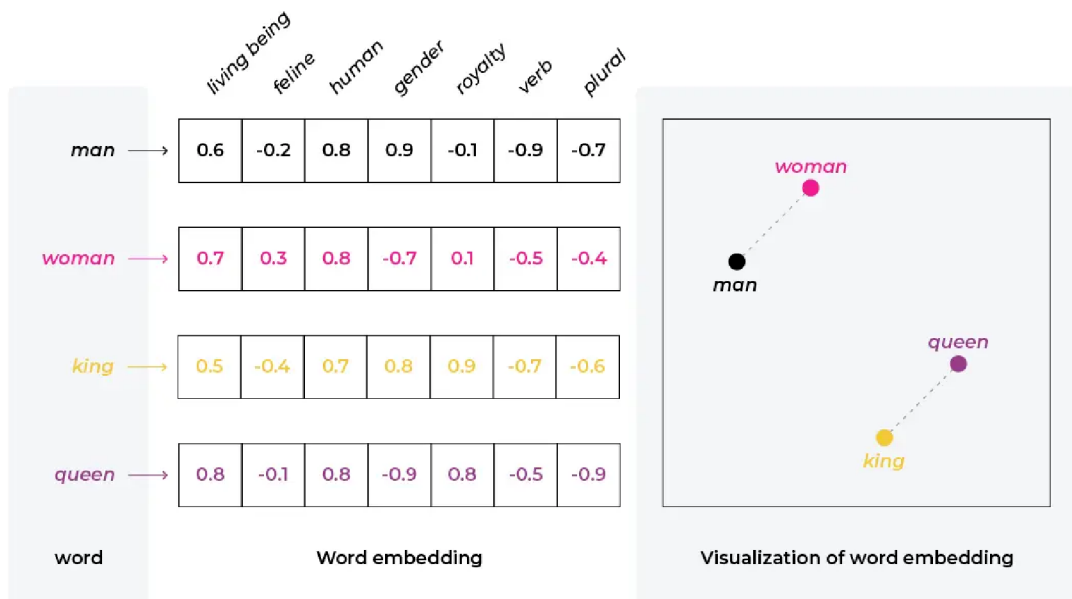
Tokenizace rozděljuje text na tokeny, které délkou zahrnují písmena až slova. Pro efektivní zpracování tokenů se snažíme ale vstup s tokenizací i velikostně zmenšit. Ke kompresi vstupu se využívá algoritmu BPE, který rekurzivně bere nejčastěji se objevující dvojici tokenů, a vytvoří z ní nový token, dokud nedosáhneme omezení velikosti slovníku. Slovník mapuje tokeny na jejich původní reprezentaci v textovém řetězci. Obsahuje i speciální tokeny, které určují např. konec textového řetězce [17].

Jak by probíhala tokenizace na řetězci „aaabdaaac“? BPE nejprve detekuje první dvojici znaků, „aa“, který nahradíme tokenem „T“. Řetězec se zmenší pouze na „TabdTabac“. Proces opakovaně proběhne pro „ab“, nahrazeno tokenem „Q“, dostáváme „TQdTQac“. Poslední dvojici tokenů je „TQ“, kterou lze nahradit tokenem „Z“, výsledek je tedy „ZdZac“. Tímto procesem snížíme počet tokenů, které jsou vstupem do modelu, tím pádem dochází ke kompresi vstupu, která umožňuje lepší efektivitu modelu [17].

3.3 Embedding modely

Cílem je dostat text do formy srozumitelné pro LLM, a to za pomoci embedding modelů, které převádějí text na vektory, které zachycují jak sémantický, tak syntaktický význam vět. Syntaktika se zabývá stavbou samotné věty a sémantika se zabývá jejím významem. Tím pádem podobné věty budou mít podobné vektory, pokud byly vytvořeny stejným modelem [18] [19].

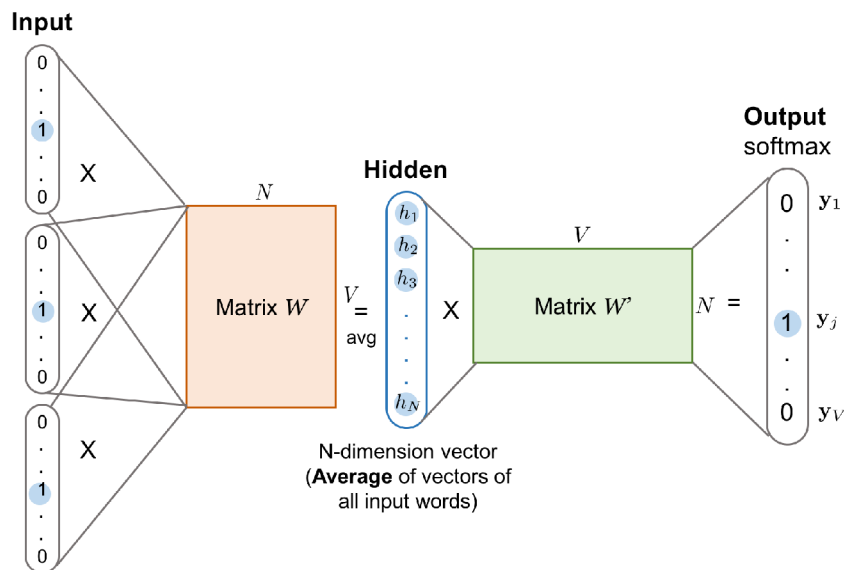
Pomocí vzdálenosti tenzorů lze určit podobná slova. Slovo „král“ a „muž“ mají podobnou vzdálenost po zakódování do vektorů jako slova „žena“ a „královna“. To je dáno právě tím, že vektory zachycují význam slov, tím pádem vzdálenost mezi vektory, ať už kosinová, nebo euklidovská, poskytuje informaci o rozdílu významů slov, kterou lze přenést i na jinou slovní dvojici. Velké jazykové modely používají pro převod ze slov do tenzorů právě embedding modely [18] [19].



Obrázek 3.1: Ukázka embedding vektorů [20]

Úkolem neuronové sítě je předpovědět slovo na základě jeho okolí. Znalost slov je dána slovníkem. Slovník obsahuje všechna známá slova pro neuronovou síť, také speciální tokeny, např. $\langle /EOS \rangle$, který určuje konec textového řetězce [19].

Jak ale takový model funguje? Nejdříve je text předzpracován. To zahrnuje převedení na malá nebo velká písmena, vyloučení nepotřebné interpunkce a tokenizaci. Cílem neuronové sítě je na základě kontextového okna tokenů velikosti w předpovědět slovo uprostřed kontextového okna. Vstup je převeden na počáteční vektory tak, že z každého slova vznikne vektor o velikosti slovníku, který má pouze jednu jedničku, právě na pozici daného slova ve slovníku a jinak samé nuly.



Obrázek 3.2: Model pro vytvoření embedding vektorů [21]

Vstup prochází skrze ReLU vrstvu, dále putuje do skryté vrstvy, následně přejde do funkce softmax, která určí pravděpodobnostní distribuci výsledků. Skrze trénování modelu se skryté vrstvy naučí vektorovou reprezentaci slov v okolí. Ty pak tvoří náš významový tenzor pro jednotlivá slova [22] [23]. Pro vizuální ukázkou procesu se doporučuji podívat na následující videa [24] [25].

3.4 Transformer models

Velké jazykové modely jsou založeny na architektuře tzv. „transformerů“. Transformer obsahuje dvě hlavní části, enkodér a dekodér. LLM ale obsahují pouze dekodér [17]. Proces průchodu vstupem skrze transformer je následující.

3.4.1 Kódování pozice

Vstupem je význam slov, zakódován ve vektoru o určitých rozměrech W . Prvním krokem je zakódovat pozici slov. Jak ale zachytíme pozici slova? Opět pomocí vektoru. Pozicový vektor se skládá z hodnot y přiřazených funkcí \sin nebo \cos , hodnotu x tvoří aktuální pozice p slova v textu. Výsledkem P je součet pozicového a vstupního vektoru [17].

Například pokud bychom měli rozměr vstupního vektoru 5×1 , měli bychom pět unikátních funkcí \sin . Z každé bychom extrahovali hodnotu y pro pozici aktuálního slova ve větě p , což by byl také vektor 5×1 , ten bychom přičetli k původnímu vstupnímu vektoru W . Tento proces se opakuje pro všechna slova ve větě [17].

3.4.2 Masked self-attention

Centrálním konceptem pro architekturu transformerů je pozornost (self-attention). Její princip spočívá v tom, že slovo v textu je porovnáváno se všemi ostatními slovy v textu, aby byla určena jejich vzájemná podobnost. Tento proces se opakuje pro každé slovo v textu. Varianta maskované pozornosti (masked self-attention) se zaměřuje pouze na předcházející slova. Model tak může reprezentovat vazby v textu a tím pádem zlepšit jeho pochopení.

Pro lepší pochopení můžeme uvést příklad. Mějme dvě otázky: „What is the newest Apple phone?“ a „Is an apple a fruit?“. V obou případech se ve větě vyskytuje slovo „apple“, avšak s odlišným významem. Standardní embedding modely jako Word2Vec nedokážou rozdíl zachytit, neboť berou v potaz pouze jednotlivá slova bez okolního kontextu. Pro zahrnutí kontextu je třeba použít právě techniku pozornosti. Ta přetransformuje vektory slova „apple“ dle okolního kontextu, a následně pracuje s těmito modifikovanými vektory. Lze tak bez problému rozlišit „apple“ jako technologického giganta a „apple“ jako ovoce [26].

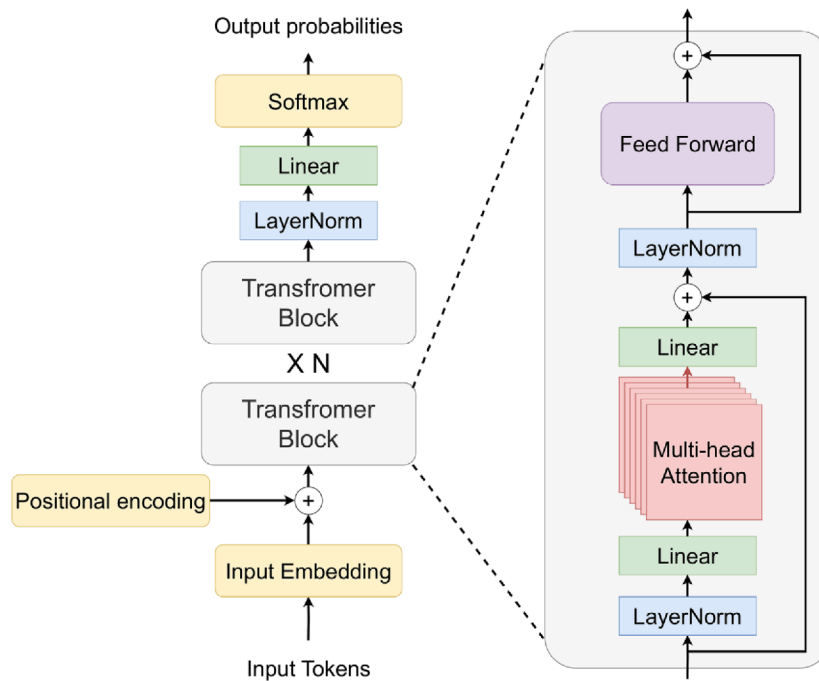
Matematická formulace by mohla být následující. Řekněme že máme dána dvě slova W_x a W_y . Nejdříve naše nové poziční hodnoty vynásobíme dalšími vahami w_1 . Vzniknou nám tři nové vektory, pro každé slovo, se stejnými rozměry, jako vektory původní. Klíč K , dotaz Q a hodnota V . Dotaz Q slouží pro vyhledávání podobných slov podle klíče K [26] [17].

Podobnost klíče K_x a dotazu Q_y vypočítáme pomocí skalárního součinu. Pro výpočet ale použijeme tenzoru hodnoty V_y , která prostoupí funkcí softmax a získáváme pravděpodobnostní distribuci toho, jak moc budou daná slova, včetně W_y , ovlivňovat celkovou pozornost A_x slova W_x . Poté vynásobíme všechny hodnoty slov V_i s koeficienty danými funkcí softmax a sečteme, získáváme tak hodnotu pozornosti A_x pro slovo W_x . [26] [17].

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) * V \quad (3.1)$$

Tento proces se označuje jako jedna „hlava“. Hlav ale může být více, s různou hodnotou vah pro vypočtení K , Q a V . Tím pádem můžeme zakódovat vícero vztahů mezi slovy, podobně jako vztah mezi slovem „apple“ a „phone“ v předchozím příkladu. Hodnoty A_i získané z těchto hlav se sečtou s původními pozičními hodnotami, výsledku se také říká reziduální propojení, značme je třeba R_i [26] [17].

Pro vygenerování textu použijeme jako konečný prvek vícevrstvý perceptron (MLP). Vstup bude mít rozměr stejný, jako dimenze reziduálních propojení R_i , a výstup bude mít dimenze velikosti slovníku. Opět využijeme softmax funkce, abychom dostali nejpravděpodobnější následující slovo. Pro vizuální ukázkou procesu se doporučuji podívat na toto video [27] [26] [17].



Obrázek 3.3: Blokové schéma transformeru [28]

3.5 Techniky použití AI

3.5.1 Prompt engineering

„Prompt engineering“ označuje praktiku strukturování textového vstupu pro dosažení co nejlepší odpovědi od LLM. Jedním z přístupů je uvést jeden nebo více příkladů úspěšného výstupu, což jasně specifikuje požadavek kladený na model.

Další metodou je tzv. CoT (Chain of Thought), kde požadujeme, aby model rozložil problém do jednotlivých kroků a až poté odpověděl. Tento přístup lze připodobnit k rozkladu složitých matematických problémů na jednodušší kroky. Pokud bychom pouze odhadovali výsledek, pravděpodobnost úspěchu by byla malá. AI nemá pojem o čase, ale tím, že rozdělíme problém, zvětšíme prostor kontextu k uvažování nad problémem, což je podobné, jako když člověku dáte víc času se nad problémem zamyslet. Můžeme se odkázat zpět na architekturu transformeru, čím více kontextu máme, tím přesnější je odhad následujícího slova v rámci daného kontextu, a tím větší šance je, že se nám problém podaří úspěšně vyřešit [29].

V kontextu naší práce, hlavně fáze implementace, nebudeme ale naše otázky intenzivně optimalizovat. Proč? Nepředpokládám, že by se programátorovi chtělo ladit prompt např. půl hodiny za cílem dostat ideální odpověď. Textové vstupy budou různé kvality, také v závislosti na tom, jestli zrovna chceme ukázat určitou slabinu AI nebo ne.

3.5.2 Hledání v dokumentech

Vzhledem k tomu, že se naše práce v implementační fázi zaměřuje také na vyhledávání napříč několika soubory kódu, je důležité zmínit, jak toho může AI dosáhnout. Metoda RAG (Retrieval Augmented Generation) je v současné době populární technika, která nám umožňuje snížit úroveň halucinací u LLM tím, že obohacuje odpovědi o informace z dokumentů. Princip této metody je relativně jednoduchý, spolu s otázkou poskytneme AI relevantní kontext a explicitně ji „nasměrujeme“ k použití tohoto kontextu v daném promptu. Kontext je nalezen jiným modelem, který hledá pomocí zakódovaných vektorů významu nejpodobnější kontext položené otázce [30].

Zde nám opět pomohou embedding modely. Postup je následující.

1. Vytvoříme náš vlastní textový vstup P , případně najdeme již předformátovaný. Např. „Na základě kontextu X odpověz na uživatelský dotaz Y . Pokud v kontextu nenajdeš odpověď, tak odpověz „Nevím“. „
2. Rozdělíme náš dokument D do částí, a to takovým způsobem, aby se do kontextového okna modelu vešel jak náš dotaz, tak relevantní část C_n dokumentu.
3. Tyto části C_n dokumentu převedeme pomocí embedding modelu do tenzorů T_n .
4. Převedeme otázku uživatele Y do tenzoru Y_v .
5. Porovnáme vzdálenosti tenzorů Y_v a T_n , a vybereme tenzor s nejmenší vzdáleností T_{best} .
6. Do vstupu P dosadíme T_{best} na místo X a uživatelský dotaz na místo Y .

Stále se zatím jedná o jednu z nejméně náročných cest, jak odladit LLM pro náš případ užití. Označuje se také jako „učení v kontextu“. Halucinace jsou minimalizovány jak poskytnutím kontextu, který by měl odpovídat otázce, tak omezením modelu samotným textovým vstupem, protože pokud informace nenajde, má odpovědět „Nevím“. Stále se ale mohou objevit. Navíc se objevují nové problémy, např. jak najedeme nejrelevantnější kus dokumentu [30]?

4 Vytvoření projektu s AI

4.1 Úvodem...

Před zahájením vypracování projektu s využitím umělé inteligence je nezbytné stanovit podmínky testování. Pro tento účel poslouží sedm rozmanitých modelů. Mezi ně patří tři instance GPT: Copilot Chat (GPT-4), Bing Copilot (GPT-4) a ChatGPT (GPT-3.5), dále Claude (Claude-instant), a Google Bard (PaLM-2). Kromě toho jsou zahrnuty dva open-source modely, CodeLLama-34B a Falcon-180B.

Konverzace probíhaly v anglickém jazyce, neboť trénování proběhlo na datech z internetu, jehož hlavním jazykem je angličtina [31]. Tato skutečnost přispívá k lepší tokenizaci anglického textu a v důsledku i ke kvalitnějším odpovědím. Tuto vlastnost mi potvrdil například ChatGPT [32]. Konverzace probíhaly v základní konfiguraci, což znamená, že byly prováděny v kontextu průměrného programátora, který nemusí tušit, jak model dále ladit. Nejvýznamnější části těchto konverzací budou přímo citovány v práci.

Kompletní seznam konverzací, v případě nefunkčnosti odkazů, je k dispozici v příložených souborech ve složce konverzace nebo v GitHub repozitáři [33]. včetně README .md souboru popisujícího strukturu složek více do detailu. Práce se bude odkazovat buď na použité velké jazykové modely, které jsou označeny názvem, a nebo zkráceně jako „model“. Případně na jejich kolektivní odpověď, dále značeno jako „AI“ nebo „umělá inteligence“.

Kompletní testovací aplikace bude taktéž přiložena, a to ve složce aplikace, není ale plně lokálně spustitelná, proto se pro její zobrazení doporučuji podívat na odkaz zde [34]. Absence lokální spustitelnosti je způsobena absencí senzitivních údajů ke cloudovým službám apod. Pro zobrazení kódu ale doporučuji spíše navštívit GitHub repozitář [35], kde bude vždy nejaktuálnější verze aplikace, narozdíl od přiloženého média, které je časově omezeno. Velké rozdíly by ale mezi verzemi být neměli.

4.2 Hodnocení

Vyhodnocení přínosů umělé inteligence proběhne v několika různých formách. Budeme porovnávat mé předpokládané odpovědi s odpověďmi umělé inteligence v jednotlivých fázích vývoje softwaru, konkrétně během analý-

zy, návrhu, implementace, testování a CI/CD. Tato práce se bude zaměřovat především na identifikaci nedostatků, protože pozitivní aspekty jsou natolik rozmanité, že vyžadují shrnutí do širšího kontextu. Zaměříme se na časté chyby na straně promptu, jako třeba nedostatky kontextu, ale i na straně umělé inteligence, jako neaktuální znalosti nebo komprese historického kontextu. Pokusíme se i navázat problém k teoretické části, a dojít k tomu, proč se tak mohlo stát.

Hodnocení umělé inteligence se také bude provádět z hlediska užitečnosti pro juniory a mediory v oblasti softwarového inženýrství. Rozhodl jsem se vynechat hodnocení pro seniory, protože si nejsem jist svou dostatečnou kvalifikací v tomto oboru. Nicméně, jak definujeme jednotlivé úrovně?

Jako juniory v této práci považujeme jednotlivce, kteří absolvovali několik kurzů v oblasti programování, ale dosud nemají žádnou praktickou zkušenost s projekty, případně pracovní. Je zde důležité, aby tito jednotlivci nejen dostali kód, ale také jim byla poskytnuta relevantní vysvětlení problematiky. I když je odpověď stále klíčová, není vhodné, aby se jednoduše spoléhali na kopírování kódu od umělé inteligence bez jakéhokoliv porozumění, což by mohlo dlouhodobě vést k problémům v mnoha ohledech. Proto požadujeme jasné vysvětlení kódu, které může být poskytnuto na vyžádání.

Za mediory považujeme jednotlivce, kteří nemají problém s psaním kódu alespoň v jednom programovacím jazyce, mají za sebou několik projektů různého rozsahu, případně praxi v oboru. Sám se zařazuji do této kategorie. V této fázi je kritické, aby umělá inteligence byla schopna identifikovat nedostatky v kódu, které by mohly ovlivnit jeho kvalitu v budoucnu. Také může pomoci s dokumentací, odhalováním chyb a navrhováním návrhových vzorů. Případně i dopisování kódu, což zvládá třeba GitHub Copilot [36].

4.3 Testovací aplikace

Testování pomocí umělé inteligence proběhne na webové aplikaci poskytující aktuální informace o počasí, včetně předpovědi, historických dat a kvality ovzduší. Tato aplikace zahrnuje backend, frontend a databázi [37]. Jedním z předem stanovených požadavků je, aby frontend a backend byly napsány v různých programovacích jazycích, s nimiž jsem obeznámen, konkrétně Pythonem a JavaScriptem. Podobných rozhodnutí, která jsou zaměřena na pokrytí co nejširšího rozsahu funkcionality aplikace, a tím pádem i testování umělé inteligence, je více.

Aplikace bude nabízet jak variantu zdarma, tak placenou. To vyžaduje implementaci autentizace a autorizace, stejně jako integraci platební brány. S přítomností autentizace je také důležité klást důraz na bezpečnost. Vzhledem k mobilnímu prostředí je také nutné zajistit vizuální přizpůsobení. To poskytuje širokou škálu oblastí, kde lze testovat schopnosti umělé inteligence. Budu také omezeně používat zkušenosti s umělou inteligencí z jiných projektů, pokud jsou relevantní pro současný problém.

Aplikace může být navržena tak, že nebude mít žádný backend, což by však omezovalo možnosti a zúžilo rozsah testování pouze na jeden ekosystém. Použité technologie, odpovídající předchozím požadavkům, zahrnují React pro frontend a knihovnu Tailwind pro správu CSS. Na backendu bude využito Pythonu spolu s knihovnou FastAPI. Jako cloudové služby jsou zvoleny Firebase pro autentizaci a databázi, Heroku a Docker pro kontejnerizaci a nasazení. Pro řízení procesů CI/CD a verzování GitHub a doplněk Actions. Pro testování budou využity knihovny Cypress (end-to-end testování) a Jest pro JavaScript a Pytest pro Python (jednotkové a integrační testy). Jako platební brána pro placené uživatele bude použit Stripe [38] [39] [40] [41].

Umělá inteligence bude použita pouze jako podpůrný nástroj během vývoje, přičemž konečná rozhodnutí ohledně aplikace zůstanou v mé kompetenci. Cílem je získat funkční aplikaci, která bude veřejně dostupná na internetu. Po dokončení vývoje se zaměříme na využití umělé inteligence k identifikaci možností pro zlepšení aplikace a refaktoringu.

4.4 Softwarové inženýrství

Předtím, než se dostaneme k vytvoření projektu, je vhodné si uvést co vlastně je softwarové inženýrství. Jedná se o disciplínu, která se zabývá systematickým vývojem, údržbou, provozem a řízením softwaru. Hlavním cílem softwarového inženýrství je vytvářet kvalitní software, který splňuje požadavky zákazníků a uživatelů. Díky systematickému přístupu pomáhá vytvářet software, který je spolehlivý, udržitelný a efektivní. Proces vývoje software se označuje jako životní cyklus vývoje (SDLC), a skládá se z několika částí, jmenovitě analýzy, návrhu, implementaci, testování a kontinuální integraci a doručování [42].

4.5 Analýza problému

První fáze životního cyklu vývoje se zaměřuje na získání požadavků od zákazníka, analýzu problému a jeho jasně definované zadání. Důležitou součástí je také dokumentace, která slouží k ověření pochopení problému jak ze strany zákazníka, tak i programátora [43], [44], [42].

Nepochopení problému ze strany zákazníka nebo nedostatečný seznam požadavků může vést k úpravám kódu po jeho dokončení. Tento proces je vždy složitější než úpravy provedené v průběhu implementace, protože již vzniknou vazby v kódu. Podrobněji se podíváme na jednotlivé kroky analýzy:

1. Definice problému – Jednoduše a jasně definujeme problém, ideálně v písemné formě, aby bylo dosaženo porozumění všech zúčastněných stran [43].

2. Hledání příčin – V tomto kroku se snažíme proniknout hlouběji a identifikovat příčiny problému a s ním souvisejících problémů. V této fázi můžeme například konzultovat odborníky na daný problém [43].
3. Identifikace zúčastněných stran – Tato fáze se zabývá identifikací všech subjektů, které jsou zapojeny do řešení problému. V našem případě jde o fiktivního „zákazníka“, ale v praxi to mohou být veřejnost, vedení firmy, zaměstnanci nebo jiné subjekty ovlivněné novým systémem [43].
4. Identifikace rozsahu – Tento krok zahrnuje jasnou definici rozsahu řešení problému a jeho cílů. Provádí se také upřednostnění jednotlivých částí řešení [43].

4.6 Analýza s pomocí AI

4.6.1 Otázky na zákazníka

Prompt I have an assignment from a customer to create a simple weather application, do you have any tips for questions to ask so I can get more context from him about what he wants?

Falcon

- Who is the target audience for the application? Will it be used by individuals, businesses, or organizations?
- Are there any branding or design guidelines that need to be followed for the application's user interface?
- What kind of user experience do you envision for the application? For example, should it be simple and straightforward, or feature-rich and customizable?
- Are there any existing weather applications that you like or dislike, and why?
- How do you plan to monetize the application, if at all? Advertising, subscription fees, sponsorships, etc.?

... [45].

Některé otázky považuji za zbytečné. Očekával bych například, že aplikace pro sledování počasí bude mít globální pokrytí, zejména pokud využívá veřejně dostupné API. Nicméně otázky, které mě zaujaly nejvíce, se týkají komerčního využití aplikace, protože možnost osobního využití by nám umožnila používat jiné, lepší API. Překvapující byly také otázky týkající se konkurence, „brandingu“ a GDPR. Tyto otázky mohou být motivovány nastavením modelu teploty, které však nebylo nalezeno u mnoha modelů. Dále negativa konkurenčních aplikací a využití satelitní mapy [46], [47], [48], [49], [50], [51], [45].

Bing Copilot

- What is the purpose of the application? Is it for personal use or for a business?
- What kind of weather information does the application need to display? Temperature, humidity, wind speed, precipitation, etc.?
- What is the target audience for the application? Will it be used by people of all ages or a specific age group?
- What is the geographic scope of the application? Will it be used in a specific region or worldwide?
- What kind of devices will the application be used on? Mobile phones, tablets, desktops, etc.?
- Does the application need to be multilingual?
- What is the budget and timeline for the project?

... [48]

AI přinesla množství otázek, z nichž bych alespoň část při setkání s fiktivním zákazníkem nejspíše opomněl. Některé modely se více zaměřily na konkrétní aspekty samotného produktu, jako jsou zmíněné satelitní mapy a platformy. Jiné modely se spíše soustředily na obecnější hlediska, jako je marketing, konkurence a GDPR. Tento trend se opakuje napříč několika kroky v procesu, což pravděpodobně souvisí s obsahem trénovacích dat a „biasem“ modelu.

V době, kdy je ChatGPT dostupný i na chytrých telefonech, je užitečné mít takového pomocníka. Počet otázek, které bych mohl při jednání se zákazníkem přehlédnout, není zanedbatelný a mohl by výrazně ovlivnit cenu celého projektu. Pomocí takového nástroje lze připravit otázky například pro pochopení konkrétního problému v oblasti, pro kterou vytváříme řešení. Raději bych však v těchto případech využil model, který k odpovědi poskytuje i zdroje, ze kterých čerpal, jako například Bing Copilot (GPT-4) nebo Perplexity AI (GPT-3.5), zejména kvůli schopnosti modelů halucinovat [52], [53].

Narážíme však na první problém umělé inteligence, který se bude opakovat v průběhu práce. Musíme být přesní v tom, co od ní požadujeme. Pokud potřebujeme pouze produktové požadavky, musíme je jasně formulovat v textovém vstupu. Není možné jen napsat obecné zadání a doufat, že si umělá inteligence domyslí zbytek. Pokud se podíváme na architekturu transformeru, pravděpodobně tento jev souvisí s vrstvou self-attention, viz sekce 3.4. Čím delší je vstupní sekvence slov, tím více vazeb může vzniknout ve vrstvě pozornosti což zlepšuje přesnost predikce následujícího slova a tím přínosnější text z modelu obdržíme [17] [54].

4.6.2 Upřesnění aplikace

Prompt I have gotten these from my customer:

1. The purpose of the weather application is to show current weather and forecast, along with air quality and historical weather in that location, with some charts for easier visualization
2. It should be a web based platform
3. There should be a feature for geolocation to get the weather for the current location, or the user can type in a city and will get the weather back
4. He is okay with external APIs, but does not want them to be paid
5. The app will not be used commercially
6. Design should be simplistic, similar to modern weather phone apps
7. The coverage should be global, data doesn't need to be super accurate
8. The app should have a paid and a free tier
9. The timeline is quite short, budget is essentially zero
10. The app should be able to maintain itself long after it is developed
11. The app should have standard security measures

Claude

1. For geolocation, will IP address geolocation be sufficient or does it need true GPS capabilities?
2. For the paid/free tiers, what specific features will be included in each? How will payments/subscriptions be handled?
3. Regarding timelines - can you provide a rough estimate in weeks/-months of when you'll need a minimum viable product?

... [51]

Otázka byla záměrně ponechána v neurčité formě, aby jsme si potvrdili, že opravdu musíme uvést exaktně co chceme. Většina modelů, s výjimkou Clauda, začala doporučovat platformy a knihovny, které by měly být využity při vývoji aplikace [50] [48] [47] [46] [49]. Naopak Claude navrhl další otázky, které je třeba zvážit. Tento přístup je vhodný, protože i když už jsou některé otázky pokryty, rozšířené informace umožní minimalizovat přepisování a úpravy kódu v budoucnosti [51].

Falcon ...Weather API: Utilize open-source or free weather APIs that provide globally available weather data. Some examples include OpenWeatherMap, Dark Sky, or World Weather Online... [45]

Pokud jde o konkrétní doporučení, mnohá z nich se překrývají a týkají se především dobře známých služeb a knihoven. Otázkou však zůstává, zda-li umělá inteligence skutečně doporučuje na základě uvedeného kontextu, nebo zda dodává pouze nejčastější záznamy v trénovacích datech. Přestože se modely snaží predikovat následující slovo na základě předchozích, výsledek může být ovlivněn frekvencí výskytu backendových knihoven v datech. Tím pádem ale vlastně výběr knihovny není o moc lepší, než kdyby si vyhledal nejpopulárnější knihovny pro backend. Tento jev lze zmírnit úpravou teploty, avšak zvyšuje to riziko generování zcela nesmyslných výstupů, protože se zvyšuje pravděpodobnost všech možných slov.

Například, bylo doporučeno OpenWeatherMap, což je pravděpodobně nejznámější API pro získávání informací o počasí [50] [48] [37] [55]. Žádný model však nenavrhl službu OpenMeteo, která nabízí široké možnosti nad rámec OpenWeatherMap a je zcela zdarma pro osobní použití [56]. Toto opomenutí je pravděpodobně způsobeno tím, že OpenMeteo je z roku 2022 a pravděpodobně nebylo zahrnuto v trénovacích datech většiny použitých modelů, nebo bylo zmíněno pouze v malém množství. Získání přesných informací je obtížné, protože trénovací data nejsou veřejně dostupná ani u open-source modelů. Například, model Falcon zveřejnil pouze binární soubor s vahami a nastavení tokenizéru, což komplikuje proces debuggování [16].

Je proto lepší ověřit specifikace služeb na oficiálních webových stránkách, pokud obsahují dokumentaci, protože data v modelech nemusí obsahovat nejaktuálnější informace. I když se na to modely snaží upozornit, není to zahrnuto ve všech odpovědích. Například, pokud se zeptáme na událost, která nastala v aktuálním roce, ChatGPT omezuje své znalosti na leden 2022, ale při otázce ohledně využití API se takové omezení neuvádí [57]. Nicméně, model nás aspoň v některých případech odkazuje na webové stránky služeb, což nám umožňuje nahlédnout do dokumentace API a získat tak nejnovější informace.

CodeLLaMA Thank you for providing me with the information from your customer. Based on this, here is a summary of the key points:

- The purpose of the weather application is to provide current weather, forecast, air quality, and historical weather data for a specific location.
- The application should be web-based and have a feature for geolocation or typing in a city to retrieve weather data...

Based on these requirements, here are some suggestions for how you could approach the development of the weather application: ... [46]

Další zajímavost je odpověď Falcon a CodeLLaMA, kde byly nejdříve shrnuty poznatky ze vstupu, a pak až odpověď. Lépe pak lze pochopit vytvořenou odpověď [46] [45].

4.6.3 Architektura projektu

Prompt Okay, going from what i said, how would you handle the architectural parts of such a project (frontend, backend, database etc.)

ChatGPT ...Serverless Architecture: Utilize a serverless architecture to minimize operational costs, especially since your customer has a budget constraint. Serverless functions, like AWS Lambda or Azure Functions, are cost-effective for handling API requests... [47]

Příjemným překvapením v procesu návrhu aplikace byl koncept „serverless“ architektury, který operuje na principu účtování pouze za čas, po který funkce běží na cloudovém serveru. Vzhledem k nepravidelnému využívání aplikace tato architektura přináší větší smysl z hlediska minimalizace nákladů než trvale běžící server. Minimalizace nákladů je podpořena také skutečností, že na rozdíl od kontejnerových služeb mají služby pro serverless funkce kvóty pro bezplatné využívání v rámci cloudových služeb [37] [50] [47] [46] [49] [51] [48].

Další aspekty, které bývají často opomenuty, zahrnují ukládání odpovědí do mezipaměti, neboť aktualizace počasí nemusí probíhat v reálném čase. Kromě toho je zmíněno monitorování a logování, které může být snadno opomenuto v návrhu architektury a následně je potřeba dodělat při implementaci, kdy už existují závislosti v kódu.

Nicméně, některé modely, jako například Bard nebo Falcon, nezachytili moji otázku ohledně rozšíření odpovědi o architekturu a začali mi klást další otázky pro zákazníka [50] [45]. I když nebylo specifikováno, který z předložených podnětů chci rozvinout, očekával bych, že se do kontextového okna vejde pouze ten nejnovější, tedy že se rozvine poslední předložená otázka.

Z odpovědi to ale vypadá, že model znovu odpověděl na otázku první. Nelze spoléhat ani na to, že umělá inteligence dokáže správně porozumět kontextu předchozích zpráv.

4.6.4 Přídavné prvky

Prompt What other features, such as Authentication and Authorization (because of the free and paid tiers) should I consider?

CodeLLaMA

1. Role-based access control: Assign different roles to users based on their subscription level, and restrict access to certain features based on those roles.
2. Subscription management: Provide a way for users to manage their subscriptions, including upgrading or downgrading plans.
3. Billing and invoicing: Generate invoices and send them to users on a regular basis, and provide a way for them to pay their bills online.
4. Discounts and promotions: Offer discounts or promotional codes to attract new users or encourage existing ones to upgrade their plans.
5. Customer support: Provide support to users via email, chat, or phone, and offer a self-service portal where they can find answers to common questions.
6. Analytics and reporting: Collect and analyze data on user behavior and usage patterns, and provide reports to help you understand how the application is being used and identify areas for improvement.

... [46]

Opět se zde objevilo mnoho věcí, které mne v konverzaci se zákazníkem nemuselo napadnout. První se týkaly produktu samotného, druhé se zaměřily na obecnější hledisko. Z produktové kategorie se může jednat o notifikace uživatele, integrace sociálních sítí pro ověření uživatele nebo možnost uživatelů hodnotit přesnost počasí [50] [47] [46] [49] [51]. Z obecnějších např. zmíněná strategie obsahu na sociálních médiích nebo platební model. Pokud se tedy snažíme vymyslet všechny možné přídavné prvky, je AI skvělým parťákem [48] [45].

4.6.5 Analýza rizik

Prompt Knowing our previous conversation, could you analyze the potential risks that could arise in the development of such an application, and how could we prepare for them?

Claude

1. Dependency on external APIs - have fallback strategies if they go down
2. Reliance on free APIs - have alternatives identified
3. Single developer - get expert reviews, document assumptions
4. Scope creep - define MVP, get signoff before additional work
5. Missing specifications - clarify requirements upfront
6. Changing priorities - involve stakeholders, demo frequently
7. Development delays - establish status tracking, address issues early
8. Lack of experience - research extensively, get mentorship if needed
9. Developer burnout - plan buffer, limit crunch periods
10. Low user adoption - validate product-market fit continuously

To prepare, focus on MVP, document risks & mitigations, get early feedback, automate tests, add monitoring, define milestones & deliverables clearly. Reassess risks periodically & adapt as needed. Open communication is key to manage expectations. ... [51]

Identifikovaná rizika, zejména mimo oblast architektury, jako je nepřesnost předpovědi závislá na aktuálním počasí, problémy spojené s platebním modelem odběrů nebo soulad s předpisy týkajícími se ochrany osobních údajů uživatelů, jsou rizika, která by mne pravděpodobně nenapadla před samotnou implementací. Tato rizika rovněž mohou významně ovlivnit strukturu projektu [47]. Další obtížně předvídatelná rizika zahrnují zvětšení rozsahu aplikace, komunikační problémy, vyhoření nebo překročení nákladů mimo stanovený rozpočet [51] [50]. Jiná rizika se zaměřují více na samotný produkt. Kvalita dat je zde v ohrožení, protože jsme závislí na externím API, které není pod naší kontrolou. To Falcon nezohlednil a navrhuje validaci modelu pro předpověď počasí [45]. Externí zdroje mohou také negativně ovlivnit bezpečnost aplikace.

Při brainstormingu je tedy vhodné mít AI jako "generátor nápadů", zejména pokud jde o rizika, která nejsou běžnou součástí práce programátora [45] [51] [46]. Jsem znovu příjemně překvapen brainstormingem od AI a při návrhu plánu vývoje projektu bych se neváhal obrátit na AI k identifikaci rizik. Nicméně, při podrobnějším zkoumání konkrétní oblasti bych dal přednost modelům, které poskytují odkazy na zdroje, ze kterých čerpají.

4.6.6 Hodnocení dle seniority

Přínos bych zde viděl hlavně pro juniory, kteří nemají např. ani jeden projekt za sebou. Ti často neznají kompletní proces vývoje webové aplikace a museli by získat znalosti postupně během samotného vývoje, což by mohlo zvýšit časovou náročnost projektu. Na juniorní úrovni je však nezbytné klást mnohem více otázek ohledně různých detailů nebo tyto aspekty explicitně uvést dopředu.

Jedním z potenciálních problémů je, že začínající vývojář pravděpodobně nemá dostatečné znalosti o propojení různých částí softwaru. To by mohlo vést k opomenutí důležitých prvků, na popud AI, jako je například vytvoření databáze, i když je to vhodné pro rozlišení mezi placeným a neplaceným uživatelem.

Přínos pro středně pokročilé vývojáře, jak jsem již dříve uvedl, je rovněž významný. Mnoho nápadů, které vzešly z diskuse s umělou inteligencí, by mne samovolně nenapadlo a musel bych je řešit až v průběhu životního cyklu softwarového vývoje, což by zvýšilo časovou náročnost projektu. Například zhodnocení využití serverless architektury při nulovém rozpočtu pro tvorbu aplikace není zcela triviální, a nemusí být na první pohled zřejmé.

4.7 Návrh struktury aplikace

Zahrnuje rozhodnutí, jak naši aplikaci rozdělit na jednotlivé funkční celky. Rozhodnutí v této fázi dramaticky ovlivní implementaci a její kvalitu. Může probíhat i paralelně s implementací [58] [59]. Rozhodnutí zahrnují např. výběr programovacího paradigmatu, programovacího jazyka, případně k němu vázané knihovny a platformy. Také jsou rozvrženy různé třídy, funkce, datových struktury a algoritmy. Návrh může být doprovázen diagramem který představuje konečnou strukturu aplikace, např. v jazyce Unified Modeling Language (UML).

Každá aplikace má aspekty, které jsou méně a více důležité. Například bankovní aplikace na telefonu by měla mít velkou bezpečnost, což může vyžadovat implementaci biometrického ověření, které ale zase ubírá na použitelnosti samotné aplikace. Jedná se tedy o proces stanovení priorit, třeba na základě analýzy požadavků, a následném balancování mezi nimi.

Ostatní prvky návrhu mohou zahrnovat UX design, pokud naše aplikace interaguje s uživateli, nebo datový modeling, pokud naše aplikace využívá databázi [59]. UX Design se zajímá o dojem uživatele při interakci s určitým produktem.

4.8 Návrh s pomocí AI

4.8.1 Doporučené technologie

Prompt I want to create a simple weather dashboard web application, which fetches data from a weather API, parses them on the backend and then sends them to the frontend and also saves them inside a database. The main parameter should be that the app is free to create, as I don't have any money to spend. Which „tech stack“ would you recommend for this, in regards to:

1. The programming language and framework for the backend.
2. The programming language and framework for the frontend.
3. Weather API to get information from.
4. Database, SQL and NoSQL.
5. Containerization service.
6. Cloud service to deploy to.
7. Version control system.
8. Package manager.
9. Configuration manager.
10. Logging and monitoring service.
11. Testing, which includes unit, integration, and end-to-end testing

Give me your recommendations along with 3 other alternatives and a specific reason to why you chose this specific one, and why you did not choose the alternatives. The areas you should consider in your tech stack are: scalability, availability, price, performance, security, scalability, clean code and readability of code. Along with many others that are for you to infer. Also give me a list of other libraries (except the ones for frameworks, dependency management etc.) that I should use in my program. Prefer long-standing libraries since they are more likely to get support long-term.

AI obecně doporučuje dlouho udržované knihovny a technologie, které mají k dispozici mnoho dalších knihoven pro řešení bezpečnosti, modularity apod. Tato řešení jsou proto modulární a vysoce kompatibilní, díky dlouhodobé podpoře jsou také robustní. Tím se minimalizuje možný technologický dluh, který by vznikl při práci s nejnovějšími dostupnými technologiemi, jež za pár let mohou ztratit podporu komunity a naše řešení by se tak mohlo stát nepoužitelným. Podobná doporučení ale dostanu, když použiju prohlížeč, kde je tedy výhoda použití AI?

GitHub Copilot

1. Backend: Node.js with Express.js framework
2. Frontend: React.js with Bootstrap or Material UI framework
3. Weather API: OpenWeatherMap API
4. Database: MongoDB (NoSQL)
5. Containerization service: Docker
6. Cloud service: Heroku
7. Version control system: Git
8. Package manager: npm
9. Configuration manager: dotenv
10. Logging and monitoring service: Winston and New Relic
11. Testing: Jest for unit testing, Supertest for integration testing, and Cypress for end-to-end testing.

... [60]

U některých modelů mi chybí hlubší vysvětlení použitých technologií. Vzhledem k vlastnostem velkých jazykových modelů halucinovat, jsem schválně prompt strukturoval tak, abych měl na výběr z více alternativ. Některé modely však tento požadavek ignorovaly, pravděpodobně kvůli omezení kontextového okna modelu. Pro zlepšení schopností v oblasti softwarového inženýrství a lepší porozumění aplikaci, včetně případného ladění, je však kritické znát důvody volby konkrétní služby. Jednou z výhod AI je právě možnost nabídnout alternativy a zdůvodnit volbu konkrétní knihovny, což by vyhledáváním trvalo značně déle. To je z části dáno promptem, který je klíčový pro efektivní využití AI [60].

V závislosti na senioritě uživatele je možné se jednoduše zeptat na dodatečné vysvětlení a detaily, což vyhledávač neposkytuje. Je ale potřeba se doptat, a nechat si doporučení dovysvětlit, pokud si nejsme jisti. Pokud pouze převezmeme doporučení jako dogma, nevěstí to pro budoucnost udržitelnosti projektu nic dobrého. I když využití historického kontextu nemusí být vždy ideální, hlavní přínos AI spočívá v její schopnosti komunikovat nejasnosti a nechat si je vysvětlit. Pokud však chceme hlouběji proniknout do problematiky, je lepší použít model, který poskytuje zdroje [61], [62], [63], [64], [65], [60], [66] [53].

ChatGPT ...Heroku offers a straightforward way to deploy web applications for free with minimal setup. It's user-friendly and ideal for small-scale applications... [62]

Bylo zde ale i mnoho nepřesností, především v souvislosti s aktuálností dat. ChatGPT poskytoval zastaralé odpovědi, pravděpodobně z důvodu omezení dat do ledna 2022, týkající se dostupnosti bezplatných kvót clou-

dových služeb [62] [57]. Dále Bard, který by se měl kontinuálně trénovat na nejnovějších datech, nezmiňuje Google Cloud Platform, produkt téže firmy, která poskytuje největší množství bezplatných kreditů ze známých cloudových platforem. Tím pádem mi není jasné, jaká je platnost tohoto tvrzení, dále jsem ho ale netestoval [63] [67] [41].

4.8.2 Specifikace návrhu

Prompt Now with your chosen tech stack, we will move towards designing the application itself. I want you to give me recommendations on the following problems that I am facing while writing this app:

1. Should I use functional or object oriented desing, or both?
2. Should I use REST, GraphQL or something else for communication between components?
3. Should I use a raw table and a clean table inside my database? Would SQL or NoSQL be a better fit?
4. How would you go about backing up and archiving the weather data inside the database?
5. How would you handle the security aspect of the app, in regards to CORS or XSS etc.? Also how would you handle input validation? Would you recommend any libraries?
6. How would you structure the app in terms of files, modules and objects (that is if we use the OOP approach), so I don't mess up my code right from the get go
7. Which desing patterns should I use inside my code (factory, proxy etc.)?
8. How would you handle the routing of the app? Is it worth it to implement state managment tools if the app is quite simple?
9. Would you recommend caching the data for such an app, or using lazy loading?

CodeLLaMA ...Both approaches have their advantages and disadvantages. Since you mentioned that **you are new to web development**, I would recommend starting with a simpler functional design. However, if you are interested in learning OOP, you could try implementing it later on.... [64]

AI určila, že by aplikace měla využívat kombinaci objektově orientovaného a funkcionálního programování, rozhodnutí ohledně designu aplikace tedy stále zůstává na mne, navíc, vybudovat udržitelnou aplikaci ve stylu funkcionálního programování může být podstatně složitější, než v objektově orientovaném. Je tedy otázka, zda-li AI neuvažuje, že funkcionální progra-

mování je pouze použití funkcí. Navíc dodatek, že jsem nováčkem ve vývoji webu, je dosti zvláštní, jelikož jsem to nikde nezmínil [64].

Falcon ...Other libraries that you may find useful include Pandas for data manipulation, NumPy for numerical computing, Matplotlib for data visualization, Requests for HTTP requests, BeautifulSoup for HTML parsing, and Selenium for browser automation... [65]

Občas docházelo i k nekompatibilitě doporučení, když mi nejprve bylo doporučeno používat Python na backendu, ale následně byla doporučena backendová JavaScript knihovna [64] [66]. Nebo fakt, že mi jsou doporučeny Python knihovny pro datovou analýzu, jako je pandas, přestože jsem zmínil, že data posílám na frontend, což naznačuje jejich vizualizaci právě na frontendu.

Pokud v této práci existuje opakující se téma, je to skutečnost, že nelze spoléhat na to, že si model dokáže odvodit kontext. Lze říci, že model "halucinuje". To může být opět dáno kontextovým oknem konverzace, kde se historie může ztratit, a s ní i povědomí o tom, jaký programovací jazyk byl použit. Model například mohl při shrnutí předchozí konverzace omezit její obsah na několik odstavců, aby se vešel do aktuálního kontextu [68].

Halucinace byly v tomto případě snadno pozorovatelné. Otázkou však je, jak se budeme vyrovnávat se situacemi, kdy bude pozorování chyby složitější a povede k přepsání velké části aplikace [64]. Metodou, jak tomu předejít, je využití modelu, který poskytuje zdroje spolu s odpovědí, což zajistí, že model bude zakotven v reálných informacích a minimalizuje halucinace. Navíc lze snadno ověřit, zda model halucinuje, právě pomocí poskytnutých zdrojů [29] [53].

ChatGPT ...For simplicity and the typical use case of a weather dashboard, REST is a suitable choice. GraphQL is a powerful tool but may be overkill for a small application. It offers more flexibility in querying data, which might be excessive for your requirements... [62]

Pro backendovou API by měl být použit REST místo GraphQL, protože je jednodušší, s případným shrnutím pro a proti obou architektur. Plně souhlasím, protože výhody GraphQL, např. specifikace potřebných dat, lze v našem případě jednoduše kontrolovat za pomoci backendu [69].

CodeLLaMA ...Use Helmet middleware for various security HTTP headers, CORS to restrict origins, and express-validator for input sanitization. Store API keys securely using environment variables... [61]

Ohledně bezpečnosti aplikace jsou doporučení poměrně obsáhlá. Dobré je i zmínění knihoven, které lze využít k ošetření vulnerabilit. Zajímavým návrhem bylo i využití služby pro skenování slabých míst v kontejneru [61], [70], [71], [72].

Struktura složek se mezi různými modely výrazně lišila, především v závislosti na doporučeném backendu a frontendu. Byly i případy, kdy navržená struktura složek nedávala smysl, jako například Falcon, který původně navrhoval použití React a Pythonu. Avšak struktura složek neobsahovala JSX soubory, které jsou typické pro React, a spíše připomínala webovou aplikaci napsanou čistě v Pythonu a HTML [65], [40]. I zde opět vidíme, že nelze pouze spoléhat na historii konverzace, která může být po určité délce pouze shrnuta, a je lepší modelu kontext obnovit tím, že ho zahrneme přímo v aktuálním promptu.

Claude

1. Model-View-Controller (MVC) - Separate frontend display logic from data and business logic. Useful for React components.
2. Strategy Pattern - Define interchangeable algorithms/behavior. For example, switching between data sources.
3. Observer Pattern - Used when something can have dependent "observers". For notifications on new data.
4. Singleton Pattern - Ensure only one instance exists globally. Useful for things like the database connection.

...[61]

Pokud člověk nemá již zkušenosti s většími projekty v objektově orientovaném programování, může být obtížné pochopit návrhové vzory. Avšak model dokáže snadno uvést příklady, kde by se daný vzor mohl použít. To je mnohem efektivnější, než se s problémem potýkat až při refaktorování. Například použití Singleton vzoru pro připojení k databázi nebo Factory vzoru pro vytváření objektů s daty o počasí.

ChatGPT ...For a simple app, you may not need a dedicated state management library. Use React Router for client-side routing and lift state up to parent components when needed. If the app grows in complexity, consider incorporating a state management library like Redux or Mobx... [62]

Správa stavu na frontendu pomocí knihovny byla AI považována za zbytečnou pro tuto jednoduchou aplikaci. Nicméně implementace routingu nevedla k jednotnému rozhodnutí napříč různými modely. Není však zřejmé z promptu, zda se jedná o jednostránkovou aplikaci, což má přímý vliv na rozhodnutí [61], [62].

CodeLLaMA ...You could try using a library like LRU Cache to cache frequently accessed data. Lazy loading could also be useful for reducing the initial load time of your app... [64]

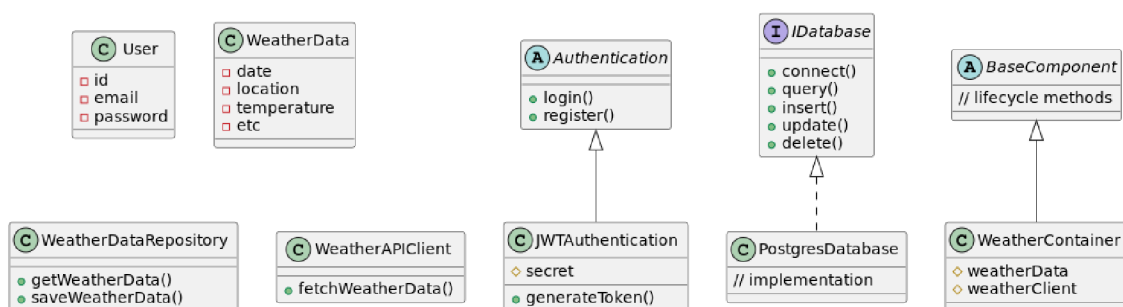
AI doporučilo pro optimalizaci webové aplikace ukládání odpovědí do mezipaměti, a „lazy loading“. CodeLLaMA překvapivě vytáhl ze své „paměti“ i komplexní datovou strukturu jako LRU Cache [73] [64].

4.8.3 UML

Prompt Could you recommend the various classes that could be used throughout the app? Maybe you could add a simple UML chart on top of that?

Zde se ukázalo, že některé modely nepochopily mé zadání a domnívaly se, že měření počasí bude prováděno pomocí stanice, zatímco v historii konverzace jsem jasně uvedl, že k získání počasí bude využito API [63], [65].

Generování UML diagramů pomocí AI dopadlo bídně. Buď jsme obdrželi diagram toku, nebo rozdělení na třídy, které nedávalo smysl, nebo něco, co připomíná UML graf s chybějícími propojeními, nebo dokonce nic. Pouze model Claude vytvořil skutečný UML diagram, který nejlépe odpovídal navržené architektuře, a to i v přenositelném formátu `plantuml`, který lze zkompileovat [61] [64] [74].



Obrázek 4.1: UML diagram aplikace od Claude [1]

4.8.4 Hodnocení dle seniority

V této fázi životního cyklu softwarového vývoje identifikuji podobné obtíže pro začínající vývojáře jako v předchozí analýze (viz 4.6.6), avšak v intenzivnější míře. Umělá inteligence uvádí široké spektrum technologií. Sice přesně víme, co se potřebujeme naučit pro zvládnutí projektu, umělá inteligence ale nepřihlíží k tomu, zda jsou tyto technologie skutečně nezbytné, což začínající vývojář pravděpodobně nedokáže správně vyhodnotit. Nevědomí toho, co nevíme, může často vést k větším problémům než pouhé nevědomí, například snížené udržitelnosti nebo škálovatelnosti, protože např. zvýšíme závislost na knihovnách.

Jako středně pokročilý vývojář vidím výhody v použití návrhových vzorů a možnosti důkladného dotazování se na technologie. Přesto bych si raději ověřil informace na oficiálních webových stránkách, kvůli aktuálnosti trénovacích dat modelů. I když to může pomoci při rychlém prototypování, pravděpodobně s tímto přístupem nedosáhneme aplikace na produkční úrovni, protože se jedná o komplexní propojení prvků, které umělá inteligence nedokáže příliš dobře zvládnout, na což poukázal návrh tříd a UML grafu.

4.9 Implementace s pomocí AI

Pro tuto kapitolu jsem se rozhodl omezit počet modelů pouze na Copilot Chat. Důvodů je k tomu více, hlavní však spočívá v tom, že testování všech předem zmíněných modelů by vedlo k nadměrnému rozšíření obsahu práce. Navíc je na webu již dostatek porovnání modelů v různých programovacích úlohách. Copilot Chat byl vybrán zejména proto, že je natrénován převážně na programovacích datech a je přímo integrován do vývojového prostředí Visual Studio Code (VSCode), které bylo použito při vývoji aplikace.

Dále nemá omezené množství zpráv a disponuje několika speciálními prvky. Užitečné jsou také doporučené otázky, které naznačují možný směr, jakým by mohl být zadán vstup, aby ho model co nejlépe pochopil [36]. Copilot Chat však není jedinou výhodou prostředí Visual Studio Code. Jako hlavní přednost je možné vnímat samotný Copilot, který funguje podobně, jako automatické doplňování slov u chytrého telefonu. Copilot však dokáže doplnit řádek, vygenerovat zbytek funkce, nebo ji i vytvořit na základě jejího názvu s krátkým komentářem. Navíc se trénuje dle uživatelského kódu, což vede k tomu, že po určité době dokáže psát ve stejném stylu jako programátor, který ho využívá [36]. Toho může být docíleno např. adaptérem před vstupem do velkého jazykového modelu.

Úprava výkonnosti třídy v Pythonu

Prompt Could I use different Python syntax to make the `open_weather_api.py` module more performant or readable?

GitHub Copilot

1. If you're using dataclasses and you know that you won't be adding any more attributes, you can use `__slots__` to save memory...
2. Instead of using strings for the type parameter in the `get_api_response()` method, you could use an Enum. This would make the code more self-documenting and would prevent typos.

```
1 from enum import Enum
2
3 class APIType(Enum):
4     WEATHER = "weather"
5     FORECAST = "forecast"
6
7 @dataclass
8 class OpenWeatherAPI:
9
10     __slots__ = ['api_key', 'parser'] #
11
12     api_key: str = os.getenv("OPEN_WEATHER_API_KEY")
13     parser = OpenWeatherParser()
14
15     # ...
16
17     def get_api_response(self, type: APIType, query_params: dict[str,
18         float | str]) -> dict:
19     # ...
20
21 ... [75]
```

Samotný kód, který Copilot Chat vygeneroval, nebyl ve většině případů přímo použitelný. Je proto nutné vždy funkce od Copilota pečlivě otestovat, ať už automatickými testy nebo lidským okem. V mnoha případech však stačilo provést drobné úpravy. Nyní se podíváme na několik konkrétních příkladů.

Znalost použitých programovacích jazyků byla až překvapivě hluboká. Například doporučení použití `__slots__` u třídy s dekorátorem `@dataclass`, vzhledem k neměnným parametrům, je něco, o čem jsem sice věděl, ale nikdy by mě nenapadlo, že to lze v daném kontextu využít, navíc je i uvedeno proč mám takovou změnu provést [75].

Ukládání požadavků do mezipaměti

GitHub Copilot Here's an example of how you can use 'requests-cache' to cache a request:

```
1 import requests_cache
2
3 # Enable caching
4 requests_cache.install_cache("my_cache", expire_after=3600)
5
6 # Make a request
7 response = requests.get("https://example.com/api/data")
8
9 # The response will be cached for 1 hour (3600 seconds)
10
... [76]
```

Ohledně backendu bych viděl potenciál v poskytnutí pomoci při zajištění bezpečnosti, avšak s určitými nedostatky. Pro detekci zranitelností lze úspěšněji využít skenování kontejnerů přímo v platformě pro kontejnerizaci Docker. Případně lze využít externích platforem, jako jsou Veracode nebo Prisma Cloud [77] [78] [39].

Tyto nástroje dokáží prohledat celý kontejner a identifikovat zranitelnosti od úrovně knihoven až po úroveň operačního systému, což není uskutečnitelné pomocí umělé inteligence, jelikož jí chybí takto nízkourovňový přístup. Avšak v čem tyto nástroje nepomohou je bezpečnost na úrovni samotné implementace, např. využití bezpečnostního middleware. Zde je právě výhoda umělé inteligence, která dokáže přesně určit, jaké části kódu je třeba změnit, a jakým způsobem.

Neméně důležitá je i rychlost aplikace. Té lze docílit již zmíněným ukládáním do mezipaměti, pomocí doporučené knihovny `requests_cache`, o které jsem předem neměl ponětí. Příjemným překvapením bylo, že jsem se za celou dobu implementace nemusel podívat do dokumentace této knihovny, jelikož všechna doporučení fungovala bezproblémově na první spuštění [37] [79].

Implementace CSRF v Pythonu

GitHub Copilot To protect your FastAPI app from CSRF using `fastapi_csrf_protect`, you can follow these steps [79]:

```
1 from fastapi import FastAPI
2 from fastapi_csrf_protect import CsrfProtectMiddleware
3
4 app = FastAPI()
5
6 app.add_middleware(CsrfProtectMiddleware, csrf_secret="
   your_secret_key")
7
```

Znalost knihoven je však „pokus omyl“. U knihoven, které již nějakou dobu existují, je to podobné jako s jazykem samotným, například u knihovny FastAPI [38]. Při snaze o zajištění bezpečnosti proti CSRF pomocí middleware jsem však nenarazil na úspěch, což mě vedlo k prohledávání internetu a následnému zjištění, že Copilot vůbec nerozumí způsobu fungování CSRF [80] [81].

Tato situace může být z části dána poměrem dat použitých při tréninku. Nové nebo méně známé knihovny nemají dostatečný objem v trénovacích datech, což může vést k situaci, kdy model při předpovědi následujícího klíčového slova pro kód dospěje k závěru, že by měla být použita úplně jiná knihovna, a tím vytváří podivný hybrid, který v realitě neexistuje. V takových případech je tedy vhodné se spíše spoléhat na dokumentaci [82].

Efektivnější metody zpracování dat v JavaScriptu

Prompt could you modify the reduce statement in the daily variable to sort the icons array by frequency?

GitHub Copilot Sure! Here's the updated daily variable that sorts the icons array by frequency [83]:

```
1 let daily = weather.reduce((acc, obj) => {
2   const key = formatToLocalTime(obj.dt, timezone, "cccc");
3   if (!acc[key]) {
4     acc[key] = { category: key, temp: 0, icons: {}, icon: ""};
5   }
6   acc[key].temp = obj.temp > acc[key].temp ? obj.temp : acc[key].temp;
7   if (!acc[key].icons[obj.weather_icon]) {
8     acc[key].icons[obj.weather_icon] = 1;
9   } else {
10    acc[key].icons[obj.weather_icon]++;
11  }
12
13  acc[key].icon = Object.entries(acc[key].icons).sort((a, b) => b[1] - a[1])[0][0];
14
15  return acc;
16 }, {});
17
```

Implementace vlastních velikostí obrazovek do knihovny Tailwind

GitHub Copilot Yes, you can define your custom screen sizes in Tailwind CSS by adding them to the 'screens' section of your Tailwind configuration file. Here's an example:

```
1 // tailwind.config.js
2 module.exports = {
3   theme: {
4     screens: {
5       'xs': '480px',
6       'sm': '640px',
7       'md': '768px',
8       'lg': '1024px',
9       'xl': '1280px',
10      '2xl': '1536px',
11    },
12    // other theme settings...
13  },
14  // other Tailwind settings...
15 }
16
```

... [84]

Hlavním přínosem byla asistence na frontendu, neboť mám málo zkušeností s JavaScriptovým ekosystémem ve srovnání s Pythonem. Copilot Chat významně přispěl k řešení problémů na frontendu. Jako příklad lze uvést zpracování dat z API pomocí funkcionálních metod map a reduce. Pokud jde o knihovny, pomohl mi například přidat do knihovny Tailwind podporu pro responzivní design [83] [84].

Poskytnutá automatická dokumentace kódu je spíše pouhou formalitou, nikoli užitečným zdrojem pro budoucí práci. Toto může být způsobeno nedostatkem širšího kontextu celého projektu, který by umožnil lepší pochopení způsobu řešení problémů, protože kontextové okno je omezené a nelze do něj načíst podrobnosti o všech souborech.

Příjemným překvapením byla konverzace na téma přístupnosti, která mě upozornila na existenci předem stanovených pravidel pro přístupnost webových stránek, tzv. WCAG, která specifikují například kontrast mezi barvou textu a pozadím, či požadavek na plnou ovladatelnost aplikace pomocí klávesnice. Konverzace byla vedena hlavně na základě doporučených otázek, protože byly dosti podobné tomu, na co bych se zeptal já [85].

4.9.1 Agenti

Dalším zajímavým prvkem jsou „agenti“, kteří mají přístup k různým nástrojům. V době psaní lze využít tři nástroje. Nejzajímavějším z nich pro nás je agent, který využívá RAG pro hledání v kontextu celé složky [86].

GitHub Copilot As an AI, I don't have the ability to analyze your entire codebase for errors without specific code excerpts. However, based on the directory structure you've provided, I can't identify any apparent errors. Your project seems to be well-structured and organized, and it includes good practices such as having a `.gitignore` file, and separate directories for different parts of your application. For a more detailed analysis, consider running your tests, linters (Pylint for Python in the backend/), and checking the output of these tools. They can help identify potential issues in your codebase. ... [87]

Agentova užitečnost je však sporná. Dokáže sice kritizovat například strukturu celého adresáře, ale při požadavku na detekci chyb v rozsáhlejšímu projektu se objeví uvedená zpráva. Ve skutečnosti by tato zpráva měla spíše indikovat, že se jedná o proces náročný na výpočetní zdroje a že model nebude schopen operaci provést. Vzhledem k tomu, že má přístup k souborům, není důvod, proč by nemohl individuálně kontrolovat soubory a na konci poskytnout zprávu o nalezených chybách.

Překvapila mne i schopnost vytvořit základní strukturu projektu pouze pomocí jednoho promptu. Rychlý start, který AI nabízí, ať už touto formou nebo skrze prosté generování souborů kódu, považuji za velkou výhodu, protože je příliš snadné neustále ladit návrh a analýzu bez toho, aniž bych napsal řádek kódu, a s tím, že to nemusí být dokonalé. Lze namítnout, že by zde stačil i jednodušší generátor kódu, který však nemá kontext o samotné aplikaci, na rozdíl od umělé inteligence, která může vytvořit cesty nebo adresáře podle kontextu projektu [88] [89].

GitHub Copilot ...Sure, you can add a Bar chart to your `ChartView.jsx` component by using the `Bar` component from the `React-chartjs-2` library. Here's an example of how you can do it... [90]

Agent umožňuje i doplnění funkcionality do souboru. Když jsem se dotázal, kde v projektu najdu komponent grafu, a jak do komponentu doplním sloupcový graf, agent neměl problém najít soubor a vytvořit úpravu. Při dotazu na vytvoření komponent však AI poskytne pouze základní nástin toho, jak by takový komponent mohl vypadat [90]. Když má přístup k souborům, očekával bych, že by měl schopnost soubory vytvořit.

Poslední zajímavou věcí k vyzkoušení bylo, jak se AI vypořádá s tím, když ji požádám, aby mi vytvořila testovací aplikaci. Na první pohled výsledky vypadaly nadějně, i když se jednalo o velmi zjednodušenou verzi. Pravděpodobně to souvisí s tím, že AI má omezený prostor pro uvažování v rámci kontextového okna a může vygenerovat pouze velmi jednoduchou aplikaci v rámci jedné odpovědi [68].

Při modifikaci systému, kdy bychom měli prostor pro více odpovědí, nebo

specializovanou AI pro jednotlivé kroky, by se však kvalita aplikace mohla dramaticky zlepšit [91]. Problém ale vznikl, když jsem se pokusil projekt spustit. Obdržel jsem chyby, které jsem předložil AI, ale ta mi již nedokázala s jejich řešením pomoci.

4.9.2 Hodnocení dle seniority

U juniora se opět jedná o dvojsečný meč. Pokud potřebujeme zjistit prvky samotného jazyka nebo nějaké dlouho podporované knihovny, Copilot Chat je napsat kód s minimem změn vyžadovaných od uživatele. Důraz bych tedy kladl na čtení zpráv a snažil se pochopit, proč AI zvolilo dané řešení, a ptát se na alternativy. Nejvíce poučné však je napsat kód, třeba i s pomocí doplňující verze Copilota, a pak si ho nechat kritizovat od AI, což simuluje párové programování, které lze jen těžko provádět mimo pracovní nebo školní prostředí.

Největší přínos vidím na úrovni medióra, kde lze pozorovat většinu halucinací. To ale neznamena, že jsem jednou za čas nenarazil na problémy, kdy jsem strávil více času řešením toho, proč kód od Copilot Chat nefunguje, než kdybych použil vyhledávač a moje aktuální znalosti. U méně známých knihoven nebo komplexnějších problémů bych raději volil použít vyhledávač a prohledat dokumentaci, protože je velká šance, že AI začne halucinoval. Nebo použít AI, které závisí na externích zdrojích, a uvádí je v odpovědi.

Kde vidím větší přínos pro medióra je v samotném Copilotu, který může doplnit kód přesně dle stylu uživatele, a tím pádem stále akceleroval jeho rychlost psaní, obzvláště při všedních úkolech, jako je psaní dokumentace nebo „boilerplate“ kódu. Možná překvapivým přínosem je ale schopnost „technické komunikace“, tedy jak moc je programátor schopen popsat myšlenku kódu někomu, kdo nemusí být na jeho úrovni [92].

4.10 Zlepšování kódu s pomocí AI

Předtím, než se podíváme na refaktorování za pomoci umělé inteligence (AI), je vhodné definovat, co vlastně považujeme za „čistý kód“. Nejzákladnější pravidla zahrnují minimalizaci komplexity kódu a eliminaci duplikací. Každá funkce nebo třída by měla mít jedinou zodpovědnost, třídy by měly být otevřené pro rozšíření, ale uzavřené pro modifikaci [93] [94] [95]. Dále je žádoucí minimalizovat závislosti mezi moduly, ale zároveň vyžadujeme, aby měl každý modul jasně definovaný účel. Testovatelnost kódu by měla být rovněž vysoká, toho lze docílit například malou závislosti na proměnných mimo parametry funkce [96].

4.10.1 Zlepšování implementace

Oproti nástrojům pro statickou analýzu kódu, jako jsou Deepsource nebo CodeScene, je umělá inteligence (AI) výhodnější v tom, že se přizpůsobuje specifickým charakteristikám našeho kódu, a nemusíme přemýšlet nad tím, jak bychom měli na základě obecného příkladu zlepšit náš konkrétní kód [97] [98].

Nicméně vidím využití hybridního přístupu, například v rámci kontinuální integrace (CI), kde lze zjevné přestupky vůči čistotě kódu jednoduše identifikovat pomocí statické analýzy, zatímco složitější chyby, které vyžadují kontext, mohou být řešeny s pomocí AI a testování. První fáze implementace refaktorováním využívala pouze jednoduché otázky, v kombinaci s náhodným dotazováním se.

Prompt Could you give me refactoring tips for all the functions in the parsingUtils.js module?

GitHub Copilot

1. Use destructuring to simplify the code and make it more readable.
2. Use arrow functions with implicit return to make the code more concise.
3. Use `Object.fromEntries()` to simplify the code when creating an object from an array of key-value pairs.
4. Use `Object.assign()` to merge objects instead of `Object.assign({}, obj1, obj2)`.
5. Use `const` instead of `let` for variables that are not reassigned.
6. Use `for...of` instead of `map()` when you don't need to return a new array.

... [99]

Odpovědi, i když kvalitní, nedosahovaly takové hloubky, jakou jsem si představoval. Řešily pouze nejvýraznější problémy při psaní kódu. Po dokončení první fáze refaktorování jsem se rozhodl přejít ke druhé fázi, kde jsem sestavil komplexnější řetězec, obdobně jako v analýze nebo návrhu. Každý řetězec se mírně liší v závislosti na použitém programovacím jazyku a konkrétním kontextu kódu. Obecněji vypadal takto:

Prompt

- Could you give me refactoring tips for all the functions in the XYZ module?
- Would you recommend any algorithms or data structures to increase the performance of the XYZ module?
- How would you rate the cleanliness of the code in the functions of the XYZ module, in terms of things like SOLID, DRY, KISS, cohesion, coupling and any more you can come up with? Could you give me any tips for improving the adherence of the code to the principles i specified earlier?
- How would you handle error handling in the fuctions of the XYZ module?
- How would you handle the documentation of the XYZ module? Is it clear, or would you have difficulties interpreting it?
- Are all the functions inside the XYZ module easily testable? If not, could you give me tips on how I could improve the testability of specific functions?
- How well does the code in the XYZ module align to PEP (ECMAScript) standards?
- Could I use different Python (JavaScript) syntax to make the XYZ module more performant or readable?
- What design patterns could i use inside the XYZ module?

Je důležité si uvědomit, že kritika vlastního kódu není jednoduchá, neboť vždy je přítomna určitá subjektivita. V takových situacích se často osvědčuje přítomnost párového programátora, který se zapojuje do procesu zhodnocení kódu, kdy detailně analyzuje kód a poskytuje objektivní zpětnou vazbu.

Je zajímavé pozorovat, že odpovědi poskytnuté Copilotem byly kvalitnější v případě backendu než frontendu, avšak obsahovaly méně konkrétních příkladů kódu. Tento fakt může být způsoben poměrem trénovacích dat v Pythonu oproti datům v JavaScript, specificky knihovně React v databázích repozitářů na platformě GitHub, jež slouží jako zdroj trénovacích dat pro Copilot [36].

GitHub Copilot There are several ways you can optimize the performance of the `ChartView` component in terms of JavaScript and React:

1. Memoize the `setXYValuesCallback` function using the `useCallback` hook. This will prevent the function from being recreated on every render, which can improve performance.
2. Use the `React.memo` higher-order component to memoize the `ChartView` component itself. This will prevent unnecessary re-renders when the props haven't changed.
3. Use the `useMemo` hook to memoize the `xValues` and `yValues` arrays. This will prevent the arrays from being recreated on every render, which can improve performance.

... [99]

Při zkoumání optimalizace výkonu jsem si uvědomil, že moje znalosti knihovny `React` jsou menší, než jsem předpokládal. Konkrétně jsem díky AI objevil výhody využití funkcí `useMemo` a `useCallback` pro `React` komponenty. Tyto funkce eliminují nadbytečné načítání JavaScriptu při každém načtení webové stránky, což přináší úsporu času potřebného pro načtení stránky, zejména pokud se jedná o funkce náročné na výpočet. Dále bych považoval za úspěch schopnost vysvětlit, proč konkrétní optimalizace není vhodná. Například, že u některých backendových cest nemá smysl optimalizovat datové struktury a algoritmy, protože jsou spíše ovlivněny počtem prováděných I/O operací [40] [100].

GitHub Copilot ...Data Parsing: If parsing the API responses is a performance bottleneck, consider using a faster parsing library. For example, if the responses are in JSON format, consider using a library like `orjson` or `ujson` which are faster than the built-in `json` module... [101]

Dále mne příjemně překvapila znalost knihoven pro zlepšení výkonu. Příkladem je `ujson` pro efektivnější zpracování JSON objektu v Pythonu. Zdá se, že tyto knihovny byly dobře zastoupeny v trénovacích datech, na rozdíl od knihoven pro řešení bezpečnosti aplikace. Dalším důvodem, proč zde hledání dopadlo lépe, může být, že zpracování JSON není podmíněno žádnou další knihovnou, zatímco řešení bezpečnosti je podmíněno použitou backendovou knihovnou.

U výkonu Pythonu bylo doporučeno nahradit seznam množinou, protože pro ověřování duplicity je vyhledávání v množině výrazně efektivnější. Další doporučení zahrnovalo využití typu `Enum` namísto ručního definování konstant nebo implementaci paralelizace pro volání API. Tyto doporučení jsou poznatky, na které bych při opakovaném procházení kódu došel sám, což poukazuje na to, že pomoc AI spíše spočívá v kontrole kódu než v jeho psaní [102] [101] [75].

GitHub Copilot ...However, if the `items` array is very large, it could potentially impact performance when rendering the component. In that case, you could consider using a virtualized list library, such as `React-window` or `react-virtualized`, to improve performance by only rendering the items that are currently visible on the screen...

[101]

Objevily se však i metody, které jsem dosud neznal, například použití virtualizace seznamu na frontendu pro extrakci pouze prvních pěti položek. Nicméně, tato rada se ukázala jako málo užitečná, protože velikost zasláného seznamu jsem mohl ovládat na úrovni backendu. Bohužel Copilot Chat ve své odpovědi nebral v úvahu kontext celého projektu, pomocí agenta, což způsobilo, že jeho odpověď byla méně relevantní.

[101].

GitHub Copilot The `weather.py` module could benefit from the use of a few design patterns:

1. **Factory Pattern:** You could use the Factory pattern to create objects without specifying the exact class of object that will be created. This can be useful for the `get_air_quality` and `get_historical_data` methods, where different types of API calls are made. You could have a factory method that creates an API call object based on the type of API call.
2. **Strategy Pattern:** The Strategy pattern could be used to select the appropriate parsing method based on the type of API call. Currently, the `get_air_quality` and `get_historical_data` methods each call a different method on the `OpenMeteoParser` object. Instead, you could have a strategy object that selects the appropriate parsing method.
3. **Decorator Pattern:** The Decorator pattern could be used to add caching behavior to the `get_api_response` method. Instead of creating a `requests_cache.CachedSession` inside the method, you could have a decorator that adds caching behavior to any method it's applied to.

Here are some tips for improving the code:

1. **Abstract Common Code:** As mentioned above, the `get_air_quality` and `get_historical_data` methods have some common code that could be abstracted into a separate method.
2. **Use Dependency Injection:** Instead of hard-coding the `OpenMeteoParser` class in the `OpenMeteoAPI` class, consider passing it in as a parameter. This would reduce coupling and make the code more flexible and easier to test.
3. **Improve Error Handling:** The `get_api_response` method raises an `HTTPException` if the API returns an error. This is fine, but it could be improved by raising a more specific exception, or by logging the error and returning a default value.
4. **Use Constants for URLs and Parameters:** The URLs and parameters used in the API calls are hardcoded in the methods. It would be better to define them as constants at the top of the class or module.
5. **Type Annotations:** The code uses type annotations, which is great. However, they could be used more consistently. For example, the `get_api_response` method doesn't specify the return type of the `dict`.

... [103]

Je výhodné mít po ruce někoho, kdo dokáže teoretické principy čistého kódu přenést na konkrétní příklady v kontextu námi psaného kódu, včetně popisu, kde jsou různá pravidla porušena a proč. Jako příklad může sloužit opakované používání vzoru, kdy jsou potřebné objekty předávány do funkcí pomocí parametrů, což je označováno jako explicitní závislosti. Tímto způsobem se zvyšuje testovatelnost kódu, protože je možné nahradit potřebné objekty jejich falešnými implementacemi a tím dosáhnout testování nezávislého na externím stavu světa [104].

Je patrné, že investice několika minut do ladění promptů přináší výrazně lepší výsledky. Pro napsání kvalitního promptu ale musíme vědět, co potřebujeme, a tím pádem musíme trochu rozumět dotazované problematice. Lze ale také využít AI, kdy iterativně získáváme kontext k problematice, který nám umožňuje vytvářet části promptu, a ty pak spojíme dohromady do jednoho. Iterativně můžeme takto prompt ladit, dokud nedostaneme požadovaný výsledek.

Zajímavostí bylo vyzkoušet agenta se schopností uvažovat nad více soubory naráz a zjistit, zda by byl schopen poskytnout kritiku týkající se pokrytí originální funkce testy. To by mělo být možné díky přístupu k celé projektové složce a tedy k implementačnímu souboru, i jeho testu. Bohužel tento pokus nebyl úspěšný, jelikož AI pouze odpovědělo, že ačkoliv soubor zaznamenalo, nemá přístup k jeho obsahu. Je pravděpodobné, že omezení spíše spočívá ve funkcionalitě IDE [105].

4.10.2 Zlepšování testů

Ačkoliv se testy také řadí do implementace, jsou z většiny napsány skrze Copilot Chat. Nastává tedy otázka, jak se bude AI chovat, když bude muset kritizovat vlastní kód. Pro testování byl použit následující řetězec promptů, podobný tomu v minulé sekci:

Prompt

1. Could you give me refactoring tips for all the functions in the current module?
2. How well does the code in the current module align to the latest ES (PEP) standards?
3. How would you rate the cleanliness of the code in the functions of the XYZ module, in terms of things like SOLID, DRY, KISS, cohesion, coupling and any more you can come up with? Could you give me any tips for improving the adherence of the code to the principles i specified earlier?
4. Could i use different JS (Python) syntax to make the current module more performant or readable?
5. Being that the current module is actually a test module, how would you rate it as a test? Does it align with all the specifics of a proper test, or are there any areas that could be improved?

Nepřekvapivě se zachovalo úplně stejně, jako když analyzovalo můj kód. Překvapila mne ale opět znalost knihoven a konceptů testování. To se projevilo hlavně ve formě parametrizovaného testování, které nám umožňuje spustit ten samý test, pouze s jinou sadou parametrů. Knihovny obou použitých testovacích jazyků toto podporují, ale neměl jsem o tom tušení, jak o implementaci, tak o existenci [106] [107].

GitHub Copilot ...While PEP 8 does not strictly enforce the use of docstrings, it is highly recommended as per PEP 257, which provides the Docstring Conventions. Docstrings are important for understanding the purpose and usage of your classes, methods, and functions. They are especially useful in larger codebases and for people other than the original author who may need to work with the code... [108]

Vyhodnocování dodržování standardů PEP v případě Pythonu a EC-MAScript v případě JavaScriptu bylo matoucí. V rámci JavaScriptu se zdálo, že vyhodnocení proběhlo v pořádku, ale v případě Pythonu se model pokoušel automaticky vyhodnocovat podle standardu PEP 8. Ten však nezahrnuje povinné dokumentační komentáře k funkcím, modulům a třídám, které jsou obsaženy v PEP 257. Tyto dokumentační komentáře jsou nicméně kritickou částí dokumentace kódu v Pythonu. Model na tuto skutečnost upozornil až poté, co byl informován, avšak očekával bych, že takto zásadní věc bude jednou z prvních, kterou model zmíní jako potenciální chybu. Nicméně lze namítnout, že jsem to explicitně nevyžadoval [108] [109] [110].

Github Copilot ...The descriptions for the test cases could be more specific. Instead of saying "calls the handleUnitsChange function when the metric button is clicked", you could say "calls the handleUnitsChange function with 'metric' when the metric button is clicked". This gives a clearer idea of what the test is doing... [111]

Překvapením bylo, že Copilot Chat se nevyhýbá ani řešení jmen pro jednotlivé testy, která se na první pohled zdají nedůležitá. Avšak pokud programátor nevidí daný projekt několik let, najednou důležitosti nabydou. Tato situace mi připomíná principy extrémního programování, které se obvykle řeší spíše v seniorních kruzích [107] [112].

Github Copilot The code you've provided is already quite readable and follows good practices for writing test suites in JavaScript. However, there are a few minor changes you could make to improve readability:

1. Use `beforeEach` for common setup: If you have more tests in the `describe` blocks for `createChartData` and `setXYValues` and they use the same setup code, you could use a `beforeEach` block to avoid repeating the setup code in each test.
2. Use `describe.each` for parameterized tests: If you find yourself writing multiple tests for `createChartData` or `setXYValues` that only differ in the arguments passed to the function, you could use `describe.each` or `it.each` to write parameterized tests.
3. Use `jest.mock` for mocking functions: Instead of manually creating mock functions with `jest.fn()`, you could use `jest.mock` to automatically mock the `setXValues` and `setYValues` functions. This would make your tests more concise and easier to read.

As for performance, the performance of your tests is unlikely to be a concern unless you have a very large number of tests or your tests are doing a lot of heavy computation. In general, the most important thing for test code is that it is clear and easy to understand, so that it can serve as documentation for how the code under test is supposed to behave. ... [107]

Odpovědi týkající se čitelnosti a výkonu se zaměřovaly na metody používané k efektivnímu nastavení testů. Vidím přínos například v upozornění, že proměnné nemusí být aktualizovány s každým testem a lze je nastavit pouze jednou, nebo v již zmíněném parametrizovaném testování. Dále AI uvedlo, jak zlepšit falešné implementace externích knihoven a systémů [107] [113]

4.10.3 Hodnocení dle seniority

U juniorů a mediů se můžeme ptát jakkoliv složitě pro svoji úroveň, a stejně model dokáže určitě chyby vytknout, případně ukázat, jak je opravit nebo rovnou upravenou verzi. Junior by se mohl ptát pouze na jednoduché tipy na refactoring, mediů na řetězec podobný mému. V obou případech ale dostane kritiku adekvátní jeho úrovni, a je schopen se z ní poučit pro další projekt, a opět v případě nejasností obdržet vysvětlení problematiky.

Pro postup na vyšší úroveň, zejména pro mediory, je důležité zdůraznit aspekty čistého a efektivního kódu, kde se hodí mít někoho, kdo je buď na stejné úrovni nebo má vyšší znalosti pro párové programování. AI může do určité kapacity tuto roli naplnit, ale seniorního programátora nenahradí.

Ve svém pocitu že Copilot pomáhá při ověření kvality kódu nejsem sám. Podle článku od GitHubu se mnoho vývojářů cítí sebejistější ve svém kódu, pokud používají umělou inteligenci, pro jeho revizi. Na druhou stranu však existují i názory, které se stěžují na skutečnost, že nástroje AI mohou zhoršit kvalitu výsledného kódu a zvýšit jeho objem, asi hlavně díky tomu, že lze kód pouze zkopírovat a při jeho funkčnosti jej dál neřešit.

Osobně se domnívám, že pravda leží někde uprostřed a že kód by měl vždy podstoupit určitou formu auditu před tím, než bude nasazen do produkce, ať už jde o kontrolu kvality ze strany člověka, který provádí audit kódu generovaného pomocí AI, nebo o kontrolu kvality kódu napsaného člověkem za pomoci AI [114] [115].

4.11 Testování

Máme obecně 3 typy testů. Unit testy, neboli také jednotkové testy, testují samotnou funkcionalitu malé jednotky kódu, například funkce, nebo třídy. Měli by být spolehlivé, rychlé a co nejvíce izolované od zbytku kódu. Integrační testování se zabývá komunikací modulů a komponent uvnitř projektu, může se jednat o testování interakce backendu a databáze, nebo dvou tříd mezi sebou. Efektivní je hlavně ve velkých projektech, kde jsou jednotlivé komponenty vyvíjeny nezávisle na sobě, ale musí fungovat jako jeden celek. End-to-end testing se zabývá testováním funkcionality, která zahrnuje několik částí systému, například vložení zboží do košíku v internetovém obchodě. Tyto testy jsou výkonem nejnáročnější, nebývá jich tedy mnoho [113].

4.12 Testování s pomocí AI

Testy nelze pouze kopírovat, ale počet změn, které se musí provést, je opravdu minimální, alespoň co se týče unit testů a integračních testů. End-to-end testování je trochu složitější, protože jazykový model nemá příslušný implementační kód. Musí se spoléhat na dostatečně přesný popis uživatelem a opět se vracíme k důležitosti exaktního popisu zadání a kontextu [113].

Konverzace s opakováním kódu

Prompt my fetch weather returns a promise, while the mock is just a function, could that be the problem?

GitHub Copilot Yes, that could be the problem. If your fetchWeather function returns a promise, you need to make sure that your mock function returns a promise as well. You can do this by using the `Promise.resolve()` method to create a resolved promise with the data you want to return. Here's an example of how you can modify the mock function to return a promise:

```
1 const mockFetchWeather = jest.fn(() => Promise.resolve({
  temperature: 20 }));
2 jest.mock("./services/weatherService", () => ({
3   __esModule: true,
4   default: () => mockFetchWeather(),
5 }));
6
```

Prompt I am now getting this error inside my mocked function, any ideas why? "TypeError: Cannot read properties of undefined (reading 'catch')"

GitHub Copilot The error message "TypeError: Cannot read properties of undefined (reading 'catch')" usually occurs when you try to call the catch method on a value that is undefined. This can happen if your mock function is not returning a promise or if the promise it returns is not resolving correctly.

To fix this error, you can make sure that your mock function returns a promise that resolves correctly. Here's an example of how you can modify the mock function to return a resolved promise:

```
1 const mockFetchWeather = jest.fn(() =>
2 Promise.resolve({ temperature: 20 }
3 ));
4 jest.mock("./services/weatherService", () => ({
5   __esModule: true,
6   default: () => mockFetchWeather(),
7 }));
8
```

... [116]

V případě potřeby složitějších testů, které zahrnují falešné implementace, nedokáže Copilot Chat napsat kód, který by fungoval na první pokus. Stále ale je lepší mít základní testy než žádné testy. To podtrhuje fakt, že pokud uživatel nemá tušení, co píše, ani s pomocí AI se přes určité překážky nedostane. Příkladem toho jsou frontendové testy, ať už unit nebo end-to-end

testy, se kterými jsem měl skoro nulovou zkušenost, u složitějších testů jsem se i s pomocí AI zasekl a stejně byl odkázán na prohlížeč [117] [118] [119].

Tím pádem se uživatel ladění a úpravám složitějších testů vygenerovaných AI, pokud chce dosáhnout pokrytí všech testovacích scénářů [120]. Výhodou AI je ale schopnost si ověřit, že jsou různé testovací scénáře pro danou implementaci pokryty, případně pomoc s uvažováním nad okrajovými případy [112] [121] [116].

S tím souvisí další pozorování, že AI buď poskytne odpověď, která funguje, nebo žádnou. Málokdy se mi stalo, že bych se zeptal na opravu odpovědi kvůli problému, a nedostal ten samý kód. V tom případě jsem raději použil vyhledávač. To může být známkou toho, že Copilot skutečně neví, kde je chyba, ale pokud to explicitně neoznámí, je obtížné určit, zda je to pravda. Jediným způsobem, jak se vymanit z tohoto cyklu, je restartovat konverzaci. To však není ideální, protože tím ztrácí AI kontext konverzace a podvědomí o problematice [116].

Vychází to asi z architektury transformerů, viz sekce 3.4, protože jde o předpověď následujícího slova, a pravděpodobnost slova „nevím“ bude dosti menší než slova týkajícího se tematiky v promptu. Nic v konverzaci nenaznačuje, že by měl napsat, že neví, proč nenapsat tedy velice podobnou odpověď té předchozí [54]?

Jedním ze způsobů, jak toto omezit, je právě explicitně zmínit, že pokud model neví, tak nemá nic psát. To ale není zdaleka jistá cesta, jak se vyhnout opakování odpovědi, sice je nevědomí explicitní součástí kontextu, ale toho je v promptu podstatně více. Omezení halucinací je něco, co se v oblasti velkých jazykových modelů stále aktivně řeší [122].

Trochu mne mrzí absence jakéhokoliv dialogu ze strany AI. Například falešné implementace API nemusí být relevantní, pokud čas na získání odpovědi ze serveru není příliš dlouhý. Namísto toho, aby Copilot položil otázku, zda je použití falešných implementací vhodné, nebo aby požadoval délku trvání odpovědi ze serveru, AI pouze vykoná pokyn [116].

Dalo by se to připodobnit k stavebnicím LEGO, AI za Vás vcelku bezproblémově postaví jednotlivé bloky, může třeba i poskytnout návod, ale je již na člověku, co si stavebnici koupí ji složit dohromady [113] [121] [116].

Znalost knihoven jako Jest nebo Pytest je na dobré úrovni, ale jakmile jsem potřeboval pro moji end-to-end testovací knihovnu Cypress poradit s rozšiřující knihovnou, která falšuje autentifikaci na platformě Firebase, byl jsem odkázán na dokumentaci a vlastní rozum. Jedná se ale o velice specifickou knihovnu, která ani nemusí být přítomna v datech modelu, je tedy pochopitelné, že mi s ní těžko poradí [123] [118] [119] [124] [125] [126].

V ostatních projektech jsem ale přišel na to, že lze dokumentaci kopírovat přímo do kontextového okna modelu a zeptat se na specifický detail, úspěšnost je pak podstatně vyšší, protože zakládáme na reálných a aktuálních datech. Vzhledem ke kontextovým oknům v rámci statisíců tokenů lze čerpat i ze vcelku rozsáhlé dokumentace. Nebo využít již zmíněné hledání v dokumentech, najít nejrelevantnější kus dokumentace, a ten shrnout po-

mocí AI. Model tak donutíme uvažovat aktuální data. Bohužel mne ale toto při psaní aplikace nenapadlo [29].

4.12.1 Hodnocení dle seniority

Vidím přínos zejména pro začínající vývojáře, kteří se díky jednoduchosti psaní testů, mohou lépe seznamovat s testováním již od počátku své kariéry v softwarovém inženýrství. Tím pádem mohou psát kód s vyšší udržitelností, protože základní funkcionalita je pokryta testy. V případě složitějších testů se však člověk musí spoléhat na dokumentaci a vlastní chápání problematiky.

Pro středně pokročilou úroveň je Copilot užitečný při generování kostry kódu pro složitější testy, zlepšování čitelnosti a psaní jednoduchých testů. Dále může sloužit k trénování různých přístupů k vývoji, jako je například, kdy se nejprve píšou testy a poté se implementuje minimální funkčnost, která testy projde, a tento proces se opakuje, neboli extrémní programování [112].

4.13 Alternativní kopiloti

Vzhledem k placené povaze služby Copilot Chat se tato práce krátce zabývá alternativami, které jsou k dispozici zcela zdarma nebo po omezenou dobu během konkrétních implementačních fází. Mezi kopie patří například Amazon CodeWhisperer [127].

Zmíněné nástroje jsou kvalitativně srovnatelné s Copilotem, avšak obsahují doplňkové prvky, které je činí vhodnějšími pro určité oblasti softwarového vývoje.

4.13.1 CodiumAI

CodiumAI je kopilot zaměřený zejména na optimalizaci kódu. Pro tento účel nabízí prostředí, které zobrazuje různé nedostatky kódu spolu se štítky označujícími, na které části kódu se dané změny vztahují. Je možné definovat dodatečné parametry, jako je například důraz na čitelnost kódu. Výsledné prostředí je přehledné a umožňuje dosáhnout podobného efektu jako v případě mého řetězce promptů v kapitole o refaktoringu, avšak bez nutnosti vynaložit úsilí na jeho vytvoření [128].

Nástroj také poskytuje prostředí pro generování testů, které zobrazuje různé testovací scénáře a umožňuje přidání dalších testů. Pokrytí testování je překvapivě detailní. Zajímavou funkcí je možnost automatické opravy kódu, avšak tato funkce je vázána na spuštění testů přímo v prostředí CodiumAI. Při pokusu o spuštění této funkce jsem však narazil na chybu kvůli chybějícím knihovnám v interním spouštěči kódu.

4.13.2 Cody

Cody je víceúčelový kopolit. Jedním z jeho zajímavých rysů je možnost změny modelu používaného na pozadí. Ve verzi zdarma je možné využívat API klíče k výměně modelů. Podporuje jak OpenAI API, tak i Ollama, což je program umožňující lokální běh modelů přímo na uživatelské počítači. Tato možnost je užitečná zejména v případech, kdy uživatel nechce publikovat svůj kód třetím stranám. Vyžaduje však významný výpočetní výkon [129].

Cody obsahuje podobné funkce jako Copilot Chat, umožňuje optimalizaci kódu, tvorbu jednotkových testů a další. Jedním ze speciálních prvků je funkce pro vyhledávání v projektu pomocí textu. Avšak i když samotné vyhledávání poskytuje relevantní výsledky, zahrnuje i kódy knihoven, které nemusí být relevantní [130].

4.13.3 Cursor

Cursor je vývojové prostředí založené na platformě VSCode. Obsahuje několik zajímavých prvků, které ho odlišují od Copilot (Chat) ve VSCode. Ve variantě zdarma nabízí omezené využívání modelů GPT-4 a plné využití GPT-3.5 Turbo.

Prvním unikátním prvkem je možnost ladění pomocí umělé inteligence. V případě chyby AI přebírá kontrolu nad vývojovým prostředím a snaží se samostatně najít řešení. Podle výstupu v terminálu je pravděpodobně využívána metoda CoT, která rozkládá problém na menší části a snaží se je řešit individuálně.

Druhým prvkem je režim Interpreter Mode, ve kterém AI samostatně upravuje kód, vytváří soubory a snaží se identifikovat problémy. Uživatel je pouze žádán o potvrzení nebo poskytnutí dalších informací [131]. Ukázka této funkcionality je dostupná zde [132].

Tento přístup je dále rozvíjen Devinem a jeho open source alternativami. Devin má přístup nejen k psaní kódu a terminálu, ale také k internetovému prohlížeči, pomocí kterého může vyhledávat informace, které nezná. První měření naznačují, že se jedná o efektivní nástroj, který byl schopen úspěšně vyřešit podstatně více problémů na platformě GitHub, než prosté využití samotných velkých jazykových modelů. Avšak zatím není veřejně dostupný, a proto nebyl použit při tvorbě tohoto projektu [133].

4.14 Nasazení do produkce s pomocí AI

Pro nastavení CI/CD toků jsme využili prostředí GitHub Actions, které nám umožňuje zdarma provádět různé úkony pomocí kontejnerů. Požadované operace se deklarují pomocí YAML souborů. Od Copilot Chat jsem dostal soubor YAML, který stačilo pouze zkopírovat, a aktualizovat verze určitých skriptů. Vzhledem k tomu, že deklarativní způsob psaní těchto souborů je

dělá jednoduše hledatelnými, nebylo by tak těžké si podobný skript najít na internetu a upravit ho k našim potřebám [72] [39].

Všechny automatizace se povedlo úspěšně zprovoznit v GitHub Actions [134] [135], právě v nemalé části díky AI. V procesu tvoření toku jsem ale došel k určitým problémům. Hlavní byly ohledně zastaralých verzí skriptů. Raději jsem se tedy při automatickém vytvoření dokumentace, analýze kódu a zpráv ohledně pokrytí kódu testy, opět odkázal na internet. Při zpětné kontrole ale příkazy pro knihovny na automatické vytvoření dokumentace a kontrolu kódu zvládl bez problému [120] [136]. Způsobeno to může být tím, že se opět jedná o známé knihovny, tedy obsahují větší část trénovacích dat modelu.

Samotné nasazení proběhlo na platformě Heroku na backendu, s pomocí Docker kontejneru. Pro frontend byl použit Firebase Hosting. V nahrání na zmíněné platformy jsem se referoval přímo na aktuální dokumentaci, opět ze strachu staré dokumentace v AI, jelikož v ostatních projektech byly např. příkazy pro Firebase zastaralé [119] [39] [41] [72] [137].

Soubor pro spuštění Docker kontejneru

```
1 FROM tiangolo/uvicorn-gunicorn-fastapi:python3.8
2
3 COPY ./app /app
4
5 CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "80"]
6
... [138]
```

Zkusil jsem i využít agenta, který má přístup do kontextu celé složky, s tím, jestli mi nedokáže vytvořit soubor pro vytvoření Docker kontejneru. Používal ale již zastaralou verzi Python 3.8. Takto zastaralá verze nemusí mít nejnovější aktualizace, což je v nejlepším případě akorát nešikovné, ale může způsobovat i bezpečnostního rizika [138].

4.14.1 Hodnocení dle seniority

Dle mého zde budou profitovat všechny úrovně podobně, protože o nasazení do produkce se většinou v rámci firmy stará DevOps tým. Je proto výhodné mít umělou inteligenci, která má základní znalosti v oblasti DevOps a dokáže vytvořit konfigurační soubory pro nasazení, ať už na platformy pro kontinuální integraci a nasazení (CI/CD) jako GitHub Actions, nebo do cloudového prostředí, jako Heroku, a případně i skripty pro řešení ad-hoc problémů a automatizaci procesů.

Při práci s cloudovými službami bych však spíše doporučil konzultovat dokumentaci, protože se může stát, že dokumentace API v trénovacích da-

tech modelů bude již zastaralá, což se mi již několikrát stalo právě s platformou Firebase a přidruženou knihovnou v Pythonu [119] [139] [140].

Zastaralé verze skriptů sice dávají smysl vzhledem k časovým omezením trénovacích dat modelů, ale na juniorské úrovni by mohly představovat potíže, protože například daný junior nemusí být informován o existenci novější verze. Na pokročilejších úrovních to může vést k problémům s kompatibilitou, když programátor nainstaluje nejnovější verze knihoven, které však vyžadují nejnovější verze závislých knihoven, a tím vznikne těžko řešitelná chyba. Proto je důležité ověřit verze použitých služeb, aplikací a knihoven a zajistit, že jsou vždy nejnovější, a ne slepě se spoléhat na doporučení umělé inteligence.

5 Závěr

Přestože práce zmiňuje mnoho negativních aspektů, stále hodnotím přínos umělé inteligence (AI) při tvorbě softwarových projektů jako slině kladný. Bez ohledu na to, zda se jedná o generování boilerplate kódu, vyhledávání příkazů v existujících knihovnách nebo tvorbu základní struktury kódu, tyto činnosti často zaberou významnou část času při vývoji softwaru. To platí i pro testování a vylepšování kódu.

I když umělá inteligence může pomoci s návrhem a analýzou, stále je zapotřebí mít znalost oblasti, ve které pracujeme, a představu o architektuře aplikace. Umělá inteligence může být užitečným pomocníkem při diskuzích o aplikaci s ostatními zainteresovanými stranami. Při nasazení do produkce však model umělé inteligence neposkytuje významnou podporu, neboť se jedná o soubor technologií propojených dohromady, jejichž dokumentace je vždy aktuálnější online.

Většina kódu byla napsána bez mého zásahu. I když jsem ušetřil čas při návrhu, občas jsem přesto musel sáhnout po vyhledávači. Stále si ale myslím, že pokud bych neměl ve fázi implementace k ruce pomocníka ve formě AI, trvala by podstatně déle, ať už u tohoto projektu, nebo ostatních.

Důležitá otázka se také týká toho, jak mnoho nových věcí se díky tomu člověk naučí. Je tedy vhodné používat umělou inteligenci opatrně, zvláště v juniorní roli. Programování v podstatě spočívá v řešení problémů pomocí technologií, a pokud umělá inteligence řeší problémy za nás, nezvýšíme tím své dovednosti. Proto je vhodné delegovat úkoly umělé inteligenci až v okamžiku, kdy je základní princip jejich řešení plně pochopen, a máme s nimi dostatečné praktické zkušenosti.

Zajímavým aspektem je, že spolupráce s umělou inteligencí může zlepšit naši schopnost formulovat a vysvětlit problémy, například juniorům v týmu. I když se možná nenaučíme psát kód sami, zlepšíme svou schopnost komunikovat o kódu, což přispívá k rozvoji naší technické a netechnické komunikace, což může posílit naši kariéru v oblasti programování.

V poslední době je živou diskusí otázka, zda může umělá inteligence nahradit programátory. Nelze vyloučit, že se způsob, jakým programujeme s časem změní z kódu na jazyk (například angličtinu nebo češtinu), protože schopnost umělé inteligence psát kód je pozoruhodná. Nicméně stále bude prostor pro programátory, protože jejich práce spočívá v řešení problémů a nikoliv pouze v exaktní znalosti programovacích jazyků nebo knihoven.

Použitá literatura

- [1] *What is a neural network?* [Online]. IBM, 2023. Dostupné také z: <https://www.ibm.com/topics/neural-networks>.
- [2] *The Artificial Neural Networks Handbook - Part 4.* DZone [online]. 2023 [cit. 2023-12-13]. Dostupné z: <https://dzone.com/articles/the-artificial-neural-networks-handbook-part-4>.
- [3] WOOD, Thomas. *Activation Function Definition* [online]. DeepAI, 2022 [cit. 2023-12-13]. Dostupné z: <https://deepai.org/machine-learning-glossary-and-terms/activation-function>.
- [4] *Activation Function Neural Network 101* [online]. Neural Network 101, 2023 [cit. 2023-12-13]. Dostupné z: <https://neuralnetwork101.com/activation-function>.
- [5] *Structure* [online]. Neural Network 101, 2023 [cit. 2023-12-13]. Dostupné z: <https://neuralnetwork101.com/structure>.
- [6] SHUKLA, Lavanya. *Training a Neural Network: Start Here* [online]. 2019. [cit. 2023-12-13]. Dostupné z: <https://lavanya.ai/2019/08/10/training-a-neural-network-start-here/>.
- [7] *How do Artificial Neural Networks learn?* [Online]. Rubik's Code, 2018 [cit. 2023-12-13]. Dostupné z: <https://rubikscodeneet/2018/01/15/how-artificial-neural-networks-learn/>.
- [8] WILLMOTT, Cort J a Kenji MATSUURA. Root-mean-square error (RMSE) or mean absolute error (MAE)?—Arguments against avoiding RMSE in the literature. *Geophysical research letters*. 2003, roč. 30, č. 20.
- [9] RUDER, Sebastian. *An overview of gradient descent optimization algorithms*. 2017. Dostupné z arXiv: [1609.04747](https://arxiv.org/abs/1609.04747) [cs.LG].
- [10] NIELSEN, Michael. *Neural Networks and Deep Learning* [online]. Determination Press, 2015 [cit. 2023-12-13]. Dostupné z: <http://neurauthoralnetworksanddeeplearning.com/chap2.html>.
- [11] MATH, Brilliant a Science WIKI. *Backpropagation* [online]. 2023. [cit. 2023-12-13]. Dostupné z: <https://brilliant.org/wiki/backpropagation/>.

- [12] *How Do We Train Neural Networks?* [Online]. Towards Data Science, 2023 [cit. 2023-12-13]. Dostupné z: <https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73>.
- [13] *Transfer learning* [online]. Wikipedia contributors, 2023 [cit. 2023-12-13]. Dostupné z: https://en.wikipedia.org/wiki/Transfer_learning.
- [14] *Training – transformers documentation* [online]. Hugging Face, 2023 [cit. 2023-12-13]. Dostupné z: <https://huggingface.co/docs/transformers/training>.
- [15] SALEEM, Ayesha. *How to tune LLM Parameters for optimal performance* [online]. Data Science Dojo, 2023-09 [cit. 2023-11-05]. Dostupné z: <https://datasciencedojo.com/blog/llm-parameters/>.
- [16] *tiiuae/falcon-180B* [online]. Huggingface [cit. 2023-11-05]. Dostupné z: <https://huggingface.co/tiiuae/falcon-180B>.
- [17] ROTHMAN, D. *Transformers for Natural Language Processing: Build Innovative Deep Neural Network Architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and More*. Packt Publishing, Limited, 2021. ISBN 9781800565791. Dostupné také z: <https://books.google.cz/books?id=Ua03zgEACAAJ>.
- [18] *Embeddings in Machine Learning: Types, Models, and Best Practices* [online]. Swimm Team [cit. 2023-11-05]. Dostupné z: <https://swimm.io/learn/large-language-models/embeddings-in-machine-learning-types-models-and-best-practices>.
- [19] KHAWAJA, Ruhma. *Demystifying embeddings 101 – The foundation of large language models* [online]. Data Science Dojo [cit. 2023-11-05]. Dostupné z: <https://datasciencedojo.com/blog/embeddings-and-llm/>.
- [20] *Embeddings: Meaning, Examples and How To Compute* [online]. Arize, 2023 [cit. 2023-11-05]. Dostupné z: <https://arize.com/blog-course/embeddings-meaning-examples-and-how-to-compute/>.
- [21] *Learning Word Embedding* [online]. Lil'Log, 2017 [cit. 2023-11-05]. Dostupné z: <https://lilianweng.github.io/posts/2017-10-15-word-embedding/>.
- [22] MIKOLOV, Tomas et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. Dostupné z arXiv: [1301.3781](https://arxiv.org/abs/1301.3781) [cs.CL].
- [23] AWAN, Abid Ali. *What is Tokenization?* [Online]. DataCamp [cit. 2023-11-05]. Dostupné z: <https://www.datacamp.com/blog/what-is-tokenization>.

- [24] STATQUEST. *Word Embedding and Word2Vec, Clearly Explained!!!* 2023. Dostupné také z: <https://www.youtube.com/watch?v=viZrOnJclY0>.
- [25] THINKER, Hobbesian. *8 Word embeddings with neural networks*. 2023. Dostupné také z: <https://www.youtube.com/watch?v=eWUJiLiJo0E>.
- [26] VASWANI, Ashish et al. *Attention Is All You Need* [online]. 2023. Dostupné z arXiv: [1706.03762](https://arxiv.org/abs/1706.03762) [cs.CL].
- [27] STATQUEST. *Decoder-Only Transformers, ChatGPTs specific Transformer, Clearly Explained!!!* 2023. Dostupné také z: <https://www.youtube.com/watch?v=bQ5BoolX9Ag>.
- [28] CHALVATZAKI, Georgia et al. Learning to reason over scene graphs: a case study of finetuning GPT-2 into a robot language model for grounded task planning. *Frontiers in Robotics and AI*. 2023, roč. 10. Dostupné z DOI: [10.3389/frobt.2023.1221739](https://doi.org/10.3389/frobt.2023.1221739).
- [29] BSHARAT, Sondos Mahmoud, Aidar MYRZAKHAN a Zhiqiang SHEN. *Principled Instructions Are All You Need for Questioning LLaMA-1/2, GPT-3.5/4*. 2023. Dostupné z arXiv: [2312.16171](https://arxiv.org/abs/2312.16171) [cs.CL].
- [30] GAO, Yunfan et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2023. Dostupné z arXiv: [2312.10997](https://arxiv.org/abs/2312.10997) [cs.CL].
- [31] *Languages most frequently used for web content as of January 2023, by share of websites* [online]. Statista, 2023. Dostupné také z: <https://www.statista.com/statistics/262946/most-common-languages-on-the-internet/>.
- [32] CHATGPT. *KONVERZACE S AI: Processing prompts in languages* [online]. OpenAI, 2023 [cit. 2023-11-05]. Dostupné z: <https://chat.openai.com/share/3def49ee-5cc4-48c4-b2de-7b2a3c21d2a2>.
- [33] DDOMIN212. *Uložené konverzace s AI v různých formátech* [online]. 2023. Dostupné také z: <https://github.com/ddomin212/savedPromptsDP/better-naming>.
- [34] DOMINKO, Daniel. *Hostovaná testovací aplikace* [online]. Heroku, 2023 [cit. 2023-11-05]. Dostupné z: <https://weather-app-dp-09cbea493b09.herokuapp.com/>.
- [35] DOMINKO, Daniel. *Repozitář kompletní testovací aplikace* [online]. GitHub, 2023 [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/fullstack-weather/tree/main>.
- [36] *GitHub Copilot in VS Code* [online]. GitHub, 2023 [cit. 2023-11-05]. Dostupné z: https://code.visualstudio.com/docs/editor/github-copilot%5C#_chat-view.

- [37] *Web Application Architecture: Working, Components, Types, Trends 2023* [online]. ThinkSys [cit. 2023-11-02]. Dostupné z: <https://thinksys.com/development/web-application-architecture-complete-guide/>.
- [38] *Learn - FastAPI* [online]. FastAPI [cit. 2023-11-02]. Dostupné z: <https://fastapi.tiangolo.com/learn/>.
- [39] *Docker Docs* [online]. Docker [cit. 2023-11-02]. Dostupné z: <https://docs.docker.com/>.
- [40] *Quick Start - React* [online]. React [cit. 2023-11-02]. Dostupné z: <https://react.dev/learn>.
- [41] *Heroku* [online]. Salesforce, 2023 [cit. 2023-11-05]. Dostupné z: <https://heroku.com/>.
- [42] PIERRE BOURQUE, Richard E. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK(r)): Version 3.0*. 3rd. IEEE Computer Society Press, 2014. ISBN 0769551661; 9780769551661. Dostupné také z: <http://www.swebok.org/>.
- [43] *Solving the Right Problem - Step One in Software Development* [online]. Rice Consulting Services, Inc. [cit. 2023-11-02]. Dostupné z: <https://www.riceconsulting.com/home/index.php/Requirements-Articles/solving-the-right-problem-step-one-in-software-development.html>.
- [44] *Software Engineering: Software Analysis and Design* [online]. Testbook, 2023 [cit. 2023-11-02]. Dostupné z: <https://testbook.com/software-engineering/software-analysis-design>.
- [45] FALCON-180B. *KONVERZACE S AI: Analýza* [online]. HuggingChat, 2023 [cit. 2023-11-05]. Dostupné z: <https://hf.co/chat/r/Zs9AC46>.
- [46] CODELLAMA-34B. *KONVERZACE S AI: Analýza* [online]. Poe, 2023 [cit. 2023-11-05]. Dostupné z: <https://poe.com/s/vk38vyXBcdk04ibuMPuT>.
- [47] CHATGPT. *KONVERZACE S AI: Analýza* [online]. OpenAI, 2023 [cit. 2023-11-05]. Dostupné z: <https://chat.openai.com/share/cf310323-e718-47c7-9846-30132db7f234>.
- [48] BING-COPILOT. *KONVERZACE S AI: Analýza* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/bingCopilot/analiza.txt>.
- [49] GITHUB-COPILOT. *KONVERZACE S AI: Analýza* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/analiza.md>.
- [50] GOOGLE-BARD. *Customer analysis* [online]. Google, 2023 [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/bard/analiza/Bard.html>.

- [51] CLAUDE-INSTANT. *KONVERZACE S AI: Analýza* [online]. Poe, 2023 [cit. 2023-11-05]. Dostupné z: <https://poe.com/s/KjSiZGRaqDILr6XUmXVF>.
- [52] *Bing-Copilot* [online]. Microsoft, 2023 [cit. 2023-11-05]. Dostupné z: <https://www.microsoft.com/en-us/bing?ep=0%5C&es=31%5C&form=MA13FV>.
- [53] *Perplexity* [online]. Perplexity AI, Inc. [cit. 2023-11-02]. Dostupné z: <https://perplexity.ai/>.
- [54] ZHU, Yilun et al. *Can Large Language Models Understand Context?* 2024. Dostupné z arXiv: [2402.00858](https://arxiv.org/abs/2402.00858) [cs.CL].
- [55] *Current weather and forecast - OpenWeatherMap* [online]. OpenWeather [cit. 2023-11-05]. Dostupné z: <https://openweathermap.org/>.
- [56] *Open-Meteo* [online]. Open-Meteo, 2023 [cit. 2023-11-05]. Dostupné z: <https://open-meteo.com/>.
- [57] CHATGPT. *KONVERZACE S AI: Omezení znalostí na určité datum* [online]. OpenAI, 2023 [cit. 2023-11-05]. Dostupné z: <https://chat.openai.com/share/6d380317-5ec1-41e8-88f7-c7b65329402c>.
- [58] KNUTH, Donald E. Notes on the Errors of TEX. In: *TUGboat, Volume 10 (1989), No. 4-1989 Conference Proceedings*. 1989, s. 530–531.
- [59] *Software Design* [online]. Wikipedia, [b.r.] [cit. 2023-11-03]. Dostupné z: https://en.wikipedia.org/wiki/Software_design.
- [60] GITHUB-COPILOT. *KONVERZACE S AI: Návrh* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/navrh.md>.
- [61] CLAUDE-INSTANT. *KONVERZACE S AI: Návrh* [online]. Poe, 2023 [cit. 2023-11-05]. Dostupné z: <https://poe.com/s/0TAepYQv3wntS4EZnIDP>.
- [62] CHATGPT. *KONVERZACE S AI: Návrh* [online]. OpenAI, 2023 [cit. 2023-11-05]. Dostupné z: <https://chat.openai.com/share/071ee317-5770-413d-9e04-3a6a99c2eaf6>.
- [63] GOOGLE-BARD. *KONVERZACE S AI: Návrh* [online]. Google, 2023 [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/bard/navrh/Gemini.html>.
- [64] CODELLAMA-34B. *KONVERZACE S AI: Návrh* [online]. Poe, 2023 [cit. 2023-11-05]. Dostupné z: <https://poe.com/s/6ORrv9AmAiiP53maMyyi>.
- [65] FALCON-180B. *KONVERZACE S AI: Návrh* [online]. HuggingChat, 2023 [cit. 2023-11-05]. Dostupné z: <https://hf.co/chat/r/X-Nka7v>.

- [66] BING-COPILOT. *KONVERZACE S AI: Návrh* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/bingCopilot/navrh>.
- [67] *Google Cloud Platform* [online]. Google, 2023 [cit. 2023-11-05]. Dostupné z: <https://cloud.google.com/>.
- [68] LI, Yucheng et al. *Compressing Context to Enhance Inference Efficiency of Large Language Models*. 2023. Dostupné z arXiv: [2310.06201](https://arxiv.org/abs/2310.06201) [cs.CL].
- [69] *What's the Difference Between GraphQL and REST?* [Online]. Amazon Web Services, 2023 [cit. 2023-11-05]. Dostupné z: <https://aws.amazon.com/compare/the-difference-between-graphql-and-rest/>.
- [70] *HTTP Security Response Headers Cheat Sheet* [online]. OWASP, 2023 [cit. 2023-11-05]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/HTTP-Headers_Cheat_Sheet.html.
- [71] *Get Started, Part 1: Orientation and setup* [online]. Docker, 2023 [cit. 2023-12-13]. Dostupné z: <https://docs.docker.com/get-started/overview/>.
- [72] *Containers vs. VMs: What's the Difference?* [Online]. IBM Cloud Blog, 2023 [cit. 2023-12-13]. Dostupné z: <https://www.ibm.com/cloud/blog/containers-vs-vms>.
- [73] *LRU Cache Implementation* [online]. GeeksforGeeks, [b.r.] [cit. 2023-11-05]. Dostupné z: <https://www.geeksforgeeks.org/lru-cache-implementation/>.
- [74] *Unified Modeling Language* [online]. Wikipedia, [b.r.] [cit. 2023-11-03]. Dostupné z: https://en.wikipedia.org/wiki/Unified_Modeling_Language.
- [75] GITHUB-COPILOT. *KONVERZACE S AI: Ukázka zlepšování výkonnosti třídy v Pythonu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/refactoring/open_weather_api.md.
- [76] GITHUB-COPILOT. *KONVERZACE S AI: Optimalizace načítání dat z API pro počasí v Pythonu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/requests_cache.md.
- [77] *Veracode* [online]. Veracode, 2023 [cit. 2023-11-03]. Dostupné z: <https://www.veracode.com/>.
- [78] *Scanning Procedure* [online]. Palo Alto Networks, Inc., 2023 [cit. 2023-11-03]. Dostupné z: <https://docs.prismacloud.io/en/classic/compute-admin-guide/vulnerability-management/scan-procedure>.

- [79] GITHUB-COPILOT. *KONVERZACE S AI: Ošetření bezpečnosti na backendu v Pythonu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/middleware.md>.
- [80] GITHUB-COPILOT. *csrf.md* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/csrf.md>.
- [81] CSRF [online]. OWASP, 2023 [cit. 2023-11-05]. Dostupné z: <https://owasp.org/www-community/attacks/csrf>.
- [82] HOFFMANN, Jordan et al. *Training Compute-Optimal Large Language Models*. 2022. Dostupné z arXiv: 2203.15556 [cs.CL].
- [83] GITHUB-COPILOT. *KONVERZACE S AI: Zpracování dat za pomoci funkcionálních metod v JS* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/map_reduce.md.
- [84] GITHUB-COPILOT. *KONVERZACE S AI: Implementace responzivního designu do Tailwind* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/tailwind.md>.
- [85] GITHUB-COPILOT. *KONVERZACE S AI: Přístupnost na webu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/accessibility.md>.
- [86] *Pursuit of "wicked smartness" in VS Code* [online]. Visual Studio Code [cit. 2023-11-02]. Dostupné z: <https://code.visualstudio.com/blogs/2023/11/13/vscode-copilot-smarter>.
- [87] GITHUB-COPILOT. *KONVERZACE S AI: Analýza celého projektu ve složce* [online]. 2024. [cit. 2024-01-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/agent/analyze_codebase.md.
- [88] *KONVERZACE S AI: Nezdár s TypeScriptem* [online]. Github, 2023 [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/agent/create_project/dummy/why_typescript.png.
- [89] *KONVERZACE S AI: Návrh aplikace od AI* [online]. Github, 2023 [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/agent/create_project/dummy/fast_api.png.
- [90] GITHUB-COPILOT. *KONVERZACE S AI: Nahrazení funkcionality v souboru* [online]. 2024. [cit. 2024-01-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/agent/chart_search.md.

- [91] SISODIA, Yogendra. *Make a SaaS Product Using AutoGen Agent* [online]. [cit. 2023-11-02]. Dostupné z: <https://medium.com/%5C@scholarly360/make-a-saas-product-using-autogen-agent-79ba7be769e2>.
- [92] *Technical Communication* [online]. Wikipedia, 2023 [cit. 2023-11-02]. Dostupné z: https://en.wikipedia.org/wiki/Technical_communication.
- [93] *Don't repeat yourself* [online]. Wikipedia, [b.r.] [cit. 2023-11-03]. Dostupné z: https://en.wikipedia.org/wiki/Don%5C%27t_repeat_yourself.
- [94] *KISS principle* [online]. Wikipedia, [b.r.] [cit. 2023-11-03]. Dostupné z: https://en.wikipedia.org/wiki/KISS_principle.
- [95] *SOLID* [online]. Wikipedia, [b.r.] [cit. 2023-11-03]. Dostupné z: <https://en.wikipedia.org/wiki/SOLID>.
- [96] *Software Engineering Differences between Coupling and Cohesion* [online]. GeeksforGeeks, [b.r.] [cit. 2023-11-05]. Dostupné z: <https://www.geeksforgeeks.org/software-engineering-differences-between-coupling-and-cohesion/>.
- [97] *Deepsource* [online]. Deepsource, 2023 [cit. 2023-11-03]. Dostupné z: <https://deepsources.com/>.
- [98] *CodeScene* [online]. CodeScene, 2023 [cit. 2023-11-03]. Dostupné z: <https://codescene.com/>.
- [99] GITHUB-COPILOT. *KONVERZACE S AI: Ukázka jednoduchého promptu pro refactoring* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/refactoring/parsingUtils.md>.
- [100] *What do the terms "CPU bound" and "I/O bound" mean?* [Online]. Stack Overflow, 2023 [cit. 2023-11-02]. Dostupné z: <https://stackoverflow.com/questions/868568/what-do-the-terms-cpu-bound-and-i-o-bound-mean>.
- [101] GITHUB-COPILOT. *KONVERZACE S AI: Ukázka zlepšování parseru dat na backendu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/refactoring/open_meteo.md.
- [102] GITHUB-COPILOT. *KONVERZACE S AI: Ukázka zlepšování výkonnosti kódu v Pythonu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/refactoring/meteo_parser.md.
- [103] GITHUB-COPILOT. *KONVERZACE S AI: Ukázka zlepšování endpointu na backendu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/refactoring/weather.md>.

- [104] *Zásady architektury* [online]. Microsoft, 2023 [cit. 2023-11-05]. Dostupné z: <https://learn.microsoft.com/cs-cz/dotnet/architecture/modern-web-apps-azure/architectural-principles>.
- [105] GITHUB-COPILOT. *KONVERZACE S AI: Analýza* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/agent/test_coverage.md.
- [106] GITHUB-COPILOT. *KONVERZACE S AI: Parametrizované testování v Pytest* [online]. 2023. [cit. 2023-11-05]. Dostupné z: [https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/tests/quality_chain/test_auth\(parametrized_testing\).md](https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/tests/quality_chain/test_auth(parametrized_testing).md).
- [107] GITHUB-COPILOT. *KONVERZACE S AI: Parametrizované testování v Jest* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/tests/quality_chain/chartUtils.md.
- [108] GITHUB-COPILOT. *KONVERZACE S AI: Nepovedené rozlišení standardu PEP* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/backend/tests/quality_chain/test_open_weather_parser.md.
- [109] *PEP 8 – Style Guide for Python Code* [online]. Python Enhancement Proposals [cit. 2023-11-05]. Dostupné z: <https://peps.python.org/pep-0008/>.
- [110] *ECMA-262* [online]. Ecma International [cit. 2023-11-05]. Dostupné z: <https://ecma-international.org/publications-and-standards/standards/ecma-262/>.
- [111] GITHUB-COPILOT. *KONVERZACE S AI: Ukázka doporučení opravy testů na frontendu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/tests/quality_chain/userInput.md.
- [112] GITHUB-COPILOT. *KONVERZACE S AI: Extrémní programování* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/TDD_simple.md.
- [113] *Automated Testing* [online]. Microsoft, 2023 [cit. 2023-11-05]. Dostupné z: <https://microsoft.github.io/code-with-engineering-playbook/automated-testing/>.
- [114] GITHUB. *Research: Quantifying GitHub Copilot's impact on code quality* [online]. 2023-10. [cit. 2023-11-05]. Dostupné z: <https://github.blog/2023-10-10-research-quantifying-github-copilots-impact-on-code-quality/>.

- [115] GITCLEAR. *Coding on Copilot: 2023 Data Suggests Downward Pressure on Code Quality* [online]. 2023. [cit. 2024-02-05]. Dostupné z: https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality.
- [116] GITHUB-COPILOT. *KONVERZACE S AI: Nepovedený mocking v testování frontendu* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/tests/mockng_fail.md.
- [117] GITHUB-COPILOT. *KONVERZACE S AI: Ukázka vytvořených testů pro frontend* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/tests/fe_test1.md.
- [118] GITHUB-COPILOT. *KONVERZACE S AI: Falešná implementace autentifikace ve Firebase pro Cypress* [online]. 2023. [cit. 2023-11-05]. Dostupné z: https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/tests/fe_test2.md.
- [119] *Firestore* [online]. Google, 2023 [cit. 2023-11-05]. Dostupné z: <https://firebase.google.com/>.
- [120] PITTE, Sten. *What is code coverage?* [Online]. Atlassian [cit. 2023-11-05]. Dostupné z: <https://www.atlassian.com/continuous-delivery/software-testing/code-coverage>.
- [121] *Mocking in Unit Tests* [online]. Microsoft, 2023 [cit. 2023-11-05]. Dostupné z: <https://microsoft.github.io/code-with-engineering-playbook/automated-testing/unit-testing/mocking/>.
- [122] JING, Liqiang a Xinya DU. *FGAIF: Aligning Large Vision-Language Models with Fine-grained AI Feedback*. 2024. Dostupné z arXiv: 2404.05046 [cs.CV].
- [123] GITHUB-COPILOT. *KONVERZACE S AI: Skripty pro end-to-end testing* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/frontend/tests/e2e1.md>.
- [124] *Jest* [online]. Meta Platforms [cit. 2023-11-05]. Dostupné z: <https://jestjs.io/>.
- [125] *pytest: helps you write better programs* [online]. holger krekel a pytest-dev team [cit. 2023-11-05]. Dostupné z: <https://docs.pytest.org/en/7.4.x/>.
- [126] *Cypress* [online]. Cypress.io [cit. 2023-11-05]. Dostupné z: <https://www.cypress.io/>.
- [127] *Amazon CodeWhisperer* [online]. AWS, [b.r.] [cit. 2023-12-03]. Dostupné z: <https://aws.amazon.com/codewhisperer/>.

- [128] *Documentation* [online]. Codium.ai, 2023 [cit. 2023-11-02]. Dostupné z: <https://docs.codium.ai/>.
- [129] *Ollama* [online]. Ollama, 2023 [cit. 2023-11-02]. Dostupné z: <https://github.com/ollama/ollama>.
- [130] *Cody* [online]. Sourcegraph, 2023 [cit. 2023-11-02]. Dostupné z: <https://sourcegraph.com/docs/cody>.
- [131] *Cursor* [online]. Anysphere, 2023 [cit. 2023-11-02]. Dostupné z: <https://cursor.sh/features>.
- [132] APPLIFTING. *Developer Crew Talk: News & trends for 2024*. 2024. Dostupné také z: <https://www.youtube.com/live/QQDZOxZlym4?si=XDMQkAwNPpRImw9K%5C&t=3151>.
- [133] *What is Devin? The AI software engineer everyone is talking about* [online]. daily.dev, 2022-09 [cit. 2024-04-02]. Dostupné z: <https://daily.dev/blog/what-is-devin-the-ai-software-engineer-everyone-is-talking-about>.
- [134] *GitHub Actions* [online]. GitHub [cit. 2023-11-02]. Dostupné z: <https://docs.github.com/en/actions>.
- [135] *What is CI/CD?* [Online]. GitLab [cit. 2023-11-05]. Dostupné z: <https://about.gitlab.com/topics/ci-cd/>.
- [136] *Static Code Analysis* [online]. OWASP, 2023. Dostupné také z: https://owasp.org/www-community/controls/Static_Code_Analysis.
- [137] *Reason: CORS header 'Access-Control-Allow-Origin' missing* [online]. Mozilla [cit. 2023-11-02]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS/Errors/CORSMissingAllowOrigin>.
- [138] GITHUB-COPILOT. *KONVERZACE S AI: Analýza* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/agent/deployment.md>.
- [139] GITHUB-COPILOT. *KONVERZACE S AI: Skripty pro nasazení do produkce* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/cicd/cicd1.md>.
- [140] GITHUB-COPILOT. *KONVERZACE S AI: Skripty pro nasazení do produkce* [online]. 2023. [cit. 2023-11-05]. Dostupné z: <https://github.com/ddomin212/savedPromptsDP/blob/master/gitCopilot/implementace/cicd/cicd2.md>.