Tampere University

Valtteri Ala-Salmi

VAPU: VERIFYING MULTI-AGENT SYSTEM FOR SOURCE CODE MODERNISATION

Master's thesis
Faculty of Information Technology and Communication Sciences
Examiners: Prof. Pekka Abrahamsson
Co-supervisor Zeeshan Rasheed
August 2025

ABSTRACT

Valtteri Ala-Salmi: VAPU: Verifying multi-agent system for source code modernisation
Master's thesis
Tampere University
Master's Programme in Information Technology
August 2025

Applications, especially ones operating on the web, are very often deprecated due to fast-moving technology. As up-keeping applications are costly and time-consuming continuous projects, many applications are left behind by their up keepers, causing challenges in security, reliability, and compatibility with other systems. Therefore, a fast and reliable system is needed to automate the needed updates for the applications.

This thesis extends a previous study of a Large Language Model (LLM)-based multi-agent pipeline system now introduced as Verifying Agent Pipeline Updater (VAPU) designed to refactor and modernise code. The thesis aims to find how different LLMs, the temperature used, and different tasks impact the performance of VAPU. Furthermore, this thesis investigates the challenges and opportunities of using a multi-agent system for updating code to provide understanding of multi-agent systems use in the practical sense.

A web legacy application was used as a case study to evaluate VAPU's performance. Six CakePHP view files from the legacy application were updated in five different LLMs at temperatures 0 and 1 with and without VAPU. VAPU's performance was further verified with 20 GitHub Python projects in different feature extension and modernisation tasks with all models, and both methods. The challenges and benefits of the multi-agent system's functionality were observed during the testing.

Comparison with different LLMs showed differences in performance. Medium-sized LLMs exceeded in smaller version update tasks, while higher-parameter models were better in more difficult tasks. With the temperature lowered to 0, VAPU improved with a reduction in errors and a higher probability of completing the requirements. Based on the evaluation of the Python files and depending on the LLM used, VAPU can achieve an improvement up to 22.5% in various code update tasks. The improvements were focused on cases with complex code.

This thesis demonstrates that a multi-agent system is capable of updating code with a possibility of higher rate of completed requirements compared to a prompt for an LLM. The thesis contributes additionally showing the benefits of lower temperature in code updating multi-agent systems. The benefits and challenges observed during the testing provide practitioners with opportunities to further improve similar systems further for updating code. VAPU provides a working foundation for practitioners for code updating multi-agent systems. The code for VAPU is publicly available at https://github.com/alasalm1/Multi-agent-pipeline

Keywords: Code Refactoring, Legacy Code, Legacy Systems, Large language model comparison, Multi-Agent System, Prompt Engineering

TIIVISTELMÄ

Valtteri Ala-Salmi: VAPU: Verifioiva moniagenttijärjestelmä lähdekoodin modernisointiin
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Elokuu 2025

Varsinkin internet-pohjaiset sovellukset ovat usein vanhentuneita, johtuen nopeasta teknologisesta kehittymisestä. Koska sovellusten ylläpitäminen on kallista ja aikaa vievää, ne jäävät päivittämättä, johtaen haasteisiin turvallisuudessa, luotettavuudessa, ja yhteensopivuudessa muihin järjestelmiin. Täten nopea ja luotettava järjestelmä on tarpeellinen automatisoimaan tarvittavat päivitykset sovelluksissa.

Tämä diplomityö laajentaa edellistä tutkimusta suuri kielimalli (LLM) -pohjaisesta moniagenttijärjestelmästä, joka on nimetty tässä työssä Verifioivaksi AgenttiPutki Uudistusjärjestelmäksi (VAPU), joka on suunniteltu päivittämään ja modernisoimaan lähdekoodia. Diplomityö pyrkii havainnoimaan kuinka eri laajat kielimallit, käytettävä lämpötila ja eri tehtävät vaikuttavat VAPU:n suorituskykyyn. Tämän lisäksi diplomityö tutkii hyötyjä ja haasteita VAPU:n käytöstä, tuottaen tietoa sen käyttömahdollisuuksista käytännössä.

Tässä työssä tapaustutkimuksena käytetään Web-pohjaista perintöjärjestelmää tarkastelemaan VAPU:n suorituskykyä. Kuusi CakePHP:n ohjelmistokehyksen tiedostoa perintöjärjestelmästä päivitettiin viidellä eri laajalla kielimallilla, nollan ja yhden mallin lämpötilalla, VAPU:lla ja ilman sitä. VAPU:n suorituskykyä tarkasteltiin tarkemmin 20:llä GitHub Python projektilla erinäisissä koodin laajennus ja modernisoinnin tehtävissä jokaisella kielimallilla ja molemmilla menetelmällä. Moniagenttijärjestelmän haasteet ja hyödyt tunnistettiin testaamisen aikana.

Vertailu suurien kielimallien kesken osoitti eroja suorituskyvyssä. Keskikokoiset suuret kielimallit olivat luotettavimpia pienemmissä version päivityksissä, kun taas laajemmat kielimallit olivat parempia vaikeammissa päivitystehtävissä. Lämpötilan ollessa 0, VAPU paransi suorituskykyä vähemmillä virheillä ja suuremmalla todennäköisyydellä suorittaa tehtävän vaatimukset. Pythontiedostojen päivityksestä saatujen tietojen perusteella, riippuen käytetystä laajasta kielimallista, VAPU voi parantaa suoritusta 22.5 % erilaisissa koodin päivityksen tehtävissä. Tämä koski varsinkin haasteellisempia lähdekoodeja.

Tämä diplomityö osoittaa, että moniagenttijärjestelmä voi päivittää lähdekoodia mahdollisesti korkeammalla todennäköisyydellä verrattuna pelkkään kehotteeseen suurelle kielimallille. Lisäksi tämä diplomityö osoittaa hyödyn käyttää alhaista lämpötilaa lähdekoodin päivittämisessä moniagenttijärjestelmällä. Havaitut hyödyt ja haasteet tarjoavat mahdollisuuden parantaa lähdekoodin päivittämiseen suunniteltuja moniagenttijärjestelmiä. VAPU tarjoaa toimivan perustan koodin päivittämiseen moniagenttijärjestelmille. VAPU:n lähdekoodi on julkisesti saatavilla osoitteesta https://github.com/alasalm1/Multi-agent-pipeline.

Avainsanat: Koodin uudelleenkirjoitus, perintökoodi, perintöjärjestelmät, Laajojen kielimallien vertailu, Multi-agentti järjestelmä, Kehotesuunnittelu

USE OF ARTIFICIAL INTELLIGENCE IN THIS WORK

Artificial intelligence (AI) has been used in generating this work:

☐ No
☒ Yes

I hereby declare, that the AI-based applications used in generating this work are as follows:

Application	**Version**
Grammarly	Free
Writeful	Overleaf
ChatGPT	GPT-5

Purpose of the use of AI

Grammarly and Writeful were used to fix grammar mistakes in the thesis, for example, punctuations and correct verb tenses. Additionally, applications were used to correct unclear world orders and unnecessary worlds in sentences. ChatGPT-5 was made to design two tables.

Parts of this work, where AI was used

Grammar and sentence correction was done for the whole thesis, where any correction was needed. The tables designed using AI are 4.12 and 4.13.

Acknowledgement of risks

I hereby acknowledge, that as the author of this work, I am fully responsible for the contents presented in this thesis. This includes the parts that were generated by an AI, in part or in their entirety. I therefore also acknowledge my responsibility in the case, where use of AI has resulted in ethical guidelines being breached.

PREFACE

This thesis was the result of many coincidences, which led to an interesting combination of two different projects at the university. I am thankful to Zeeshan Rasheed for the feedback and help during this thesis work. I also want to thank Pekka Abrahamsson for accepting and coordinating this work. Last, I want to thank Juhani Norri for the electronic dictionary used in this thesis.

in Tampere, 24th August 2025

Valtteri Ala-Salmi

CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

$\mathbf{in_i}$	Intermediate token
T	LLM temperature
$\mathbf{y_i}$	Output token
T^u	Used LLM temperature
μ	mean
σ	standard deviation
ALOC	Avarage lines of code
BERT	Bidirectional Encoder Representations from Transformers
CC	cyclomatic complexity
CoT	Chain-Of-Thought
CWE	Common Weakness Enumeration
EDP	Electronic Dictionary Project
FFN	Feed-Forward Network
GPT	Generative Pre-trained Transformer
LDB	Large Language Debugger
LLM	Large Language Model
LOC	Lines of code
LPW	Large language model programming workflow
OD	Operation difficulty
ORM	Object–relational mapping
OSL	One-Shot Learning
RF	Replaced functions
RQ	Research Question
SLM	Small Language Model
VAPU	Verifying Agent Pipeline Updater
xLTM	extended Long Short- Term Memory
ZSL	Zero-Shot Learning

1. INTRODUCTION

Large Language Model (LLM)-based multi-agent systems have been implemented lately as a way to automatize code generation from a given requirements [1–3]. LLMs have improved rapidly as their architecture is scalable [4] and as more efficient reasoning models are introduced [5]. As such, more complex multi-agent systems are also able to be made with more complex use cases. One possible area in software engineering is to utilize multi-agent systems for updating legacy applications by refactoring their code. The Internet contains a large number of web applications that contain deprecated components, which cause risks in a fast-developing environment [6, 7]. According to Demir *et al.* [7], an estimated 95% of the 5.6 million web applications analysed have a deprecated component.

Upkeeping an application takes notable resources for a company. Estimates related to code maintenance are between 40-90% of the companies' resources [8, 9]. As such, the technical debt easily accumulates, increasing the risk that the application becomes a legacy system. The owners of the application are usually stuck with the application due to the business model, and a change requires an expensive investment from the company [10]. The hesitation due to costly and possible cost-infective operation leads to problems in compatibility [11], safety [12], and usability [10] of deprecated applications. Therefore, a multi-agent system could provide a solution to automatize parts of the code maintenance, which would reduce the costs and making companies more acceptable to upkeeping their applications.

This thesis extends the work of Ala-Salmi *et al*. [13], where a multi-agent system pipeline was proposed to update legacy web applications. In this paper, the system is properly named Verifying Agent Pipeline Updater (VAPU). The system testing was extended from one to five LLMs by validating and verifying its functionality. VAPU was first verified using a deprecated electronic dictionary from which six view files were updated using the system. For comparison, the view files were updated with Zero-Shot Learning (ZSL) and One-Shot Learning (OSL) prompts. Additionally, the tests were repeated in all models with the temperature lowered to zero to investigate whether lowering the temperature to give more likely answers makes an LLM or VAPU better functioning. Then VAPU was validated using open source GitHub projects in 20 different updating scenarios again compared to using the selected LLMs alone. VAPU is made publicly available at [14].

1.1 Research Questions, Objectives and the Aims

The aim of this thesis is to study the capabilities of an LLM-based multi-agent system when updating existing code. Therefore, the scope of the aim is defined to cover LLM-based multi-agent systems which are able to generate or modify code. This thesis's first objective is to recognize how a multi-agent system can benefit programmers in code updating tasks, especially in legacy systems, and what are the challenges encountered with such systems. The second objective is to understand the different variables which impact a multi-agent system's capabilities in the code updating process. The following four research questions are proposed for this study based on these objectives:

RQ1. *How a different LLM affect code updating and a multi-agent system performance?*

The goal of ***RQ1*** is to analyse whether a different LLM affects the code-producing qualities of a code-generating multi-agent system. This includes, for example, the number of errors in the code and the amount of code produced for a certain task.

RQ2. *How changing the LLM temperature affects code updating quality and a multi-agent systems performance?*

The goal of ***RQ2*** is to analyse how a different temperature of an LLM affects a multi-agent systems performance in the code generation. Similar to ***RQ1*** different performance-related qualities like number of errors are included in the question's scope.

RQ3. *What are the challenges in a code generating multi-agent systems?*

The goal of ***RQ3*** is to recognize challenges that were encountered during the verification and validation of a code-generating multi-agent system. Any challenge that negatively impacts the performance of a multi-agent system is included in the scope of this RQ.

RQ4. *How a multi-agent system improves code generation?*

The goal of ***RQ4*** is to find how a multi-agent system improves code generation. The question is answered by how the code is improved with different metrics and comparing them to the alternative methods.

1.2 Thesis Structure

The remaining thesis is organized as follows: Chapter 2 introduces the necessary concepts for understanding LLM's functionality and how token output is calculated from the distribution with the help of a temperature parameter. Additionally, the background study reviews necessary problems with updating legacy systems and how multi-agent systems can solve them based on their performance found in the literature. In Chapter 3, present and justify the system used in this thesis. Additionally, resources are given for the evaluation of the system and the metrics used to validate the proposed system. Chapter 4

discusses the results of the system evaluation and validation with different LLMs, legacy applications, and methods. The implications of these results are discussed in Chapter 5, which provides answers to the research questions of this thesis. In the final chapter 6, the arguments and findings of the thesis are summarized.

2. BACKGROUND

This chapter discusses background studies related to legacy applications and how LLMs and multi-agent systems might be able to solve them. First, the chapter presents LLMs and the necessary technological background behind them. It includes prompts, LLM temperature, and limitations of LLMs, for example, hallucinations. Then, this chapter discusses existing code-generating LLM-based systems and how challenges in LLM code generation and especially legacy application updates can be improved using LLM and multi-agent systems.

2.1 Large Language Models

Large Language Model (LLM) is a language model with a higher parameter size compared to Small Language Models (SLM). In the literature, the parameter size required for LLM varies, for example, above 10 billion [15] or above 100 billion parameters [16]. LLMs as language models in general are used to make a prediction of the next string based on a certain input [17]. The output string is called a token, which in LLMs are short strings that should be able to create any symbol sequence required with a maximized computing cost [18]. As a predicting model, the output of LLM is therefore a vector of token probabilities where, for example, the most likely token is used to predict the next token for the string sequence. Despite LLMs commonly predict only one token at a time, there have been investigation to predict multiple tokens at the same time [19]. In Gloeckle *et al.* [19], predicting four tokens simultaneously improved coding metrics and made an LLM inference speed three times faster.

Modern implementations of LLM's architecture are usually transformer-based [20, 21]. The transformer is a network architecture introduced in Vasmani *et al.* [22] to implement machine learning models in parallel with multiple GPUs, allowing larger models to be trained and run in a reasonable time duration. Lately, research has also been conducted on other possible implementations of LLMs like extended Long Short-Term Memory (xLSTM) with promising results in perplexity values compared to transformer-based LLMs [23]. Transformers in full form consist of an encoder and a decoder which are stacks of identical layers containing multi-head attention and Feed-Forward Networks (FFNs) [22]. However, with popular LLMs encoder and decoder are often used alone in the architecture

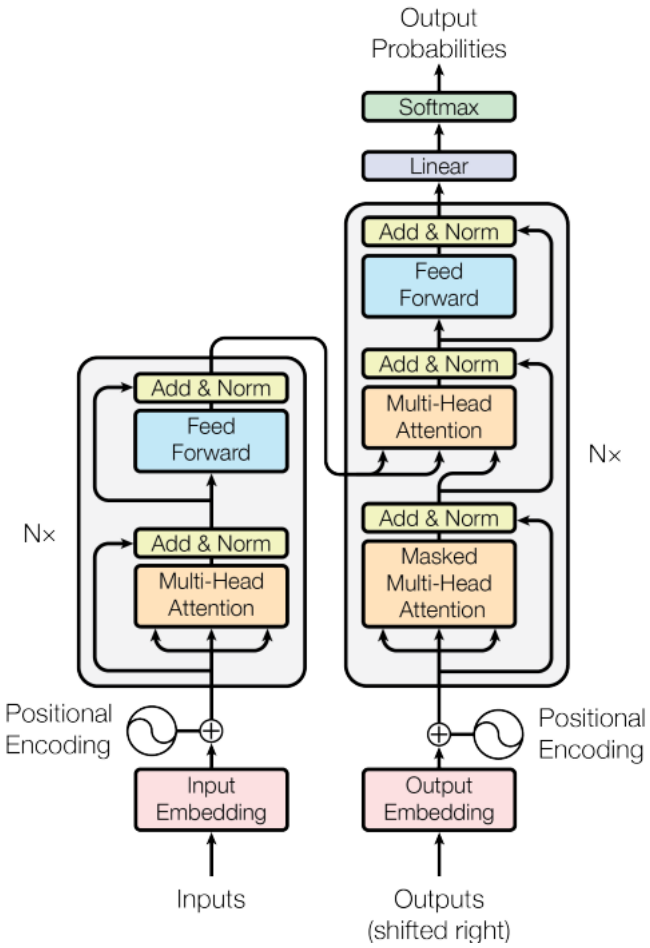


Figure 2.1. *Transformer architecture in LLM's reproduced from [22]*

[20, 21]. As such, LLMs are often separated into decoder models, for example, Generative Pre-trained Transformers (GPTs) [20] and encoder models such as Bidirectional Encoder Representations from Transformers (BERTs) [21].

The encoder is responsible for mapping the input to continuous vector representations, which are then sent to the decoder where the input and output embedding are attended to create a weighted sum as a vector [22]. The attention function in a transformer is done in parallel in different dimensions, which is called a multi-head attention. This allows the computing cost to be the same as using a single attention to the whole vector space [22]. This is then fed to FFNs, which are applied position-wise in transformers. After layers of FFNs and multi-head attentions in the decoder, the resulting outcome is linearly transformed to create logit scores for each token which can be written as $logit(p) = ln(\frac{p}{1-p})$. The scores are then normalized with a softmax function, which transforms logit into strength differences between the token scores, turning them into probabilities between them [24]. The transformer architecture containing both the encoder and the decoder is fully illustrated in Figure 2.1.

2.1.1 Prompt

A prompt is an input of a symbolic sequence that contains a task which in LLM executes and outputs as a chain of tokens. As found in Section 2.1, each output token is the result of a stochastic calculation. As a result, LLM creates a distribution of possible outputs as a chain rule from previous output tokens, input, and the given task [17]. The factorized joint probability of the output is seen in equation 2.1 where *y* is an output token.

$$p(output|input, task) = \prod_{i=1}^{n} p\left(y_i \mid y_1, \ldots, y_{i-1}, input, task\right) \tag{2.1}$$

From the outputs returned, a certain percentage will be desired outputs that fulfil the task as intended by the user. Therefore, the distribution of possible outputs allows it to be used as an evaluation of different metrics that measure the quality of LLM and used techniques by calculating the probability of desired outputs [25]. The manipulation of a prompt to get a desired result with higher probability is called prompt engineering. Using different techniques, the prompt is adjusted to a certain task, which highly improves its performance when done correctly. Different techniques have been invented to divide a prompt into smaller steps with different decomposing methods to create a process flow for the correct answer [26].

One example of process flow in prompting is to make an LLM simulate the thinking process. Chain-of-thought (CoT) is a prompting technique first described by Wei *et al*. [27]. It is designed to mimic a thought process by dividing problem-solving into steps, which allows an LLM model to make deductions in the calculated answer. It can be achieved, for example, by a few-shot prompt where the LLM answer to a question is taught to contain intermediate steps [27]. Combined with reinforcement learning, LLM can be trained to do this automatically, which models are called reasoning models [5].

Less complex techniques are also used to improve an output in prompts. A prompt can contain examples of the desired output for the LLM to improve the output probability of the desired format. If a prompt contains no examples for the LLM it is considered as Zero-Shot Learning (ZSL). If a prompt contains one example, it is classified as One-Shot Learning (OSL). With more examples, it is considered as a few-shot Learning (FSL). [28] With OSL and FSL prompt it is important to keep the examples semantically correct, as any wrong interpretation can lower the success rate as found in Reynolds & McDonell [29].

2.1.2 Temperature

One parameter to control the randomness of token generation from all possible outputs is called temperature *T*. Temperature is used in a decision function softmax in the most popular variation that utilizes the Boltzmann distribution [30]. Softmax with Boltzmann distribution in equation 2.2 controls the probability of y_i from all possible output tokens. When the temperature is lowered, the probability of the most likely tokens increases while the less likely tokens decrease [30]. As such, when the temperature is closer to zero, it will eventually return only the most likely token, making the decision deterministic. With enough temperature, the returned token will then become completely random from the available tokens.

$$P(y_i) = \frac{\exp\left(\frac{y_i}{T}\right)}{\sum_j \exp\left(\frac{y_j}{T}\right)}, T > 0 \tag{2.2}$$

When used in LLM, based on Equation 2.1, we can see that the token output equation transforms into a chain of Boltzmann distributions in Equation 2.3. Therefore, a change in temperature will cumulatively impact the probability of each generated token. As such, the temperature parameter will impact the creativity of the entire output, giving higher probabilities for more unlikely answers [31].

$$\prod_{i=1}^{n} p\left(\frac{\exp\left(\frac{y_i}{T}\right)}{\sum_j \exp\left(\frac{y_j}{T}\right)} \mid \frac{\exp\left(\frac{y_1}{T}\right)}{\sum_j \exp\left(\frac{y_j}{T}\right)}, \ldots, \frac{\exp\left(\frac{y_i-1}{T}\right)}{\sum_j \exp\left(\frac{y_j}{T}\right)}, input, task\right) \tag{2.3}$$

From the literature, some studies have tested the capabilities of the temperature parameter. In Renze & Guven [32], the impact of changing the temperature of LLM was studied. The study focused on different temperatures between 0.0 and 1.0 and tested different LLMs with multi-choice questions with varying prompt techniques. The study did not find statistical differences in performance in this temperature range between LLMs or prompt techniques. It was also found that increasing temperature above 1.0 strongly weakens the problem solving capabilities of LLM, eventually going statistically choosing randomly with a questionnaire at a value of 1.4 [32].

In Peeperkorn *et al*. [33] the temperature was evaluated up to 2.0 in four aspects novelty, typicality, cohesion, and coherence. The evaluation was done as a questionnaire for 36 participants asked to rate LLM-generated stories at different temperatures. Based on the evaluation the story novelty had a slight correlation with a trade-off in coherence. No correlation was measured in typicality and cohesion. The study came to the conclusion that temperature is not very fitting as a creativity parameter [33].

2.1.3 Limitations of LLMs

LLM has multiple features that affect its capability of processing tokens and generating the resulting probability vector of tokens. This subsection lists three recognized limitations of LLM performance found in the literature. The first is the number of parameters used in the model which must be scaled with the dataset and the model's training time to lower its test loss [4]. The training data and the model size in the parameters must be around the same size to allow the LLM's test loss to lower and its performance to improve as stated in scaling laws [4, 34]. With enough parameters and when scaling laws are obeyed, the LLMs' accuracy increases in tasks [35]. In reverse, this means that a low enough parameter size limits LLMs' ability to provide accurate results acting as a limitation for smaller LLMs.

The second recognized limitation is the context length, which tells how many tokens a model can take into consideration to calculate the next token. For example, if the maximum token length is 128K of tokens, any token over it will not be taken into consideration for the possibility vector of the next token. The token sequence size of a testing dataset will additionally impact LLM's performance when the given input is longer than the trained size [36]. Additionally, the way LLM different context lengths are distributed matters. An *et al.* [37] discussed that the positioning distribution in LLMs is often too much left due to training data favouring shorter sequences. Due to this, LLMs preferred only tokens in the beginning and performed poorly for longer sequences [37]. As such, context length limits not only the amount of information LLM can process but also the context size of data training, and its distribution impacts the model's performance.

As LLM returns the distribution of outputs to an input and the task by Equation 2.1, some of the returned instances are not desired outputs. This leads to a phenomenon called hallucination. Hallucination can mathematically be defined as in Xu *et al.* [38] as $h^{[i]}(s) \neq f(s)$, where f is a ground truth function that is only correct with any output string $s, s \in \mathbf{S}$. $h^{[i]}(s)$ means a hallucinating LLM h producing different results for the correct function $f(s)$ with any run where $i \in \mathbb{N}$ [38]. Hallucinations generated by LLM can be categorized into two main types, intrinsic and extrinsic hallucinations. Intrinsic hallucination is a type of hallucination that contradicts verifiable, often user-given source information from a given data, like summaries [39]. Extrinsic hallucinations are LLM output that cannot be verified from any source material, which could be a claim that is not mentioned in the source material [39]. This means that an extrinsic claim requires an external source to verify $h^{[i]}(s) = f(s)$, unlike an intrinsic claim, which can be verified from the source material.

Hallucination is seen as an inevitable part of LLMs, meaning that any LLM will hallucinate with a certain probability [38, 40]. This is caused, for example, by modern LLMs encountering NP-complete problems while LLM is run in polynomial time [38] or lacking the ability to deterministically find the right information [40]. Even if LLM could be able

to solve such problems, it could not learn all the data [40] or solve every problem [38], therefore, leading to hallucinating information. This is also highlighted by equation 2.3 as having temperature in an LLM leads to any outcome having a certain possibility which includes hallucinations as all output tokens have a non-zero possibility. With a deterministic LLM, this will happen if the most likely probability output contains a hallucination, as it is the only outcome.

Table 2.1. *Found limitations of LLM*

Limitation	Affects	References
Parameter size	Improves LLMs performance if scaling laws are obeyed	[4, 34, 35]
Context length	Size of input, training data should include whole length distribution	[36, 37]
Hallucination	Probability of LLM generating correct information	[38, 40]

2.2 LLM-based Systems for Generating Code

LLM-based multi-agent system is a framework of LLM agents working towards a certain goal. Each agent has a certain role and each of them is trained with role-related instructions using prompt engineering [41]. One prompt engineering technique used by a multi-agent system is called a persona pattern. In persona pattern, LLM is instructed to behave in a certain role, which improves the probability of output of which LLM is instructed to act [42]. Multi-agent systems can be divided into code-generating and non-generating systems [2]. In this thesis, the focus is on code-generating systems. This section discusses four LLM-based systems found in the literature. The main features of the systems and the collected results were examined. The results of studies are HumanEval benchmark values introduced by Chen *et al*. [25] used as a coding metric for Python files.

Shinn *et al*. [43] proposed a agent system called Reflexion. In the Reflexion system, an agent is connected to an evaluator component, which gives a numeric score based on the agent's action. The numeric score was then added to a self-reflection component, which combined with an external feedback produced a verbal analysis, which was then stored in the agent's memory for the next action [43]. Reflexion therefore forms a system where the agent improves if the self-reflection of the system succeeds. Using Reflexion, the HumanEval pass was measured as 91.0 % in GPT-4 [43].

Huang *et al*. [2] proposed a multi-agent system called Agentcoder designed to test produced code. Agentcoder consists of three agents where the first agent is tasked to produce code, the second agent to create test cases for the code, and the last agent to execute the test cases. If the test cases fail, the system returns back to the programmer agent until all the tests are completed correctly. As such, Agentcoder can return output only after all test cases are completed after loopbacks in the system. In the HumanEval

benchmark, Agentcoder obtained a result of 96.3% score using GPT-4 [2].

Zhong *et al*. [44] created an alternative code testing framework called Large Language Debugger (LDB). In LDB the code provided to the system is divided into control blocks which are tested during the program's execution, which is then verified with LLM. LDB takes into account the intermediate values of the program during testing, providing insight into errors that occur during the execution of the programs [44]. With GPT 3.5 turbo LDB had a value of 79,9% in HumanEval, but when the debugger was run with GPT 3.5 turbo and the code originally generated with GPT-4, the result obtained in HumanEval was 89.6% [44]. With GPT-4o, the LDB combined with Reflexion had metric elevated up to 98.2 % [44].

Lei *et al*. [45] introduced a system called large language model programming workflow (LPW). The system is made as a pipeline where different steps of code generation are put into components including parts from previous multi-agent systems, like runtime information of programming execution from LDB [45]. The pipeline itself was divided into two parts, with the first part generating a solution to a programming problem and the second part generating code from that solution. LPW had a value of 89.0% with GPT-3.5 turbo and 98.1% with GPT-4o [45].

Based on results from the literature, the systems that utilize techniques like feedback loop and code testing will improve changes of correct code being output by the LLM. The performance of the systems using HumanEval is visible in Table 2.2. From the table, it can be seen that each system provided an improvement to the HumanEval pass value compared to the base model. It is also noteworthy that the highest value measured was with the combination of two systems, which could indicate possible cumulative improvement of different LLM-based systems. It is important to note that the difference in the base model values can vary notably between studies, as seen in Table 2.2. This could be explained by different versions of the models, as the GPT models are updated frequently [46] and the fact that test sets have been passed once, which causes deviation of the results due to the nature of LLM [28].

2.3 Challenges in Code Generation using LLM

Generating code using LLM has challenges due to the nature of LLM working with probabilities as discussed in Section 2.1 and its limitations discussed in Subsection 2.1.3. Different studies exist that focus on the existing challenges of the code generated by LLM. Both circumstances that cause problems in the code and the problems are included as challenges in the section. This section introduces three studies found in the literature that focus on the challenges found in code generation. Based on the findings, the following challenges were found:

Table 2.2. *LLM-based systems performance compared to baseline LLM models with HumanEval metric with Pass@1*

System	Model	HumanEval (%)	Mean difference (%)	Reference
-	GPT-3.5 turbo	57.3, 73.8, 74.4	-	[2, 44, 45]
AgentCoder	GPT-3.5 turbo	79.9	11.4	[2]
LDB	GPT-3.5 turbo	82.9	14.4	[44]
-	GPT-4	67.2, 80.1, 87.2	-	[2, 43, 44]
LPW	GPT-3.5 turbo	89.0	20.5	[45]
LDB (GPT 3.5)	GPT-4	89.6	11.4	[44]
-	GPT-4o	90.2, 91.5	-	[45, 47]
Reflexion	GPT-4	91.0	12.8	[43]
AgentCoder	GPT-4	96.3	18.1	[2]
LPW	GPT-4o	98.1	7.3	[45]
LDB & Reflexion	GPT-4o	98.2	7.4	[44]

- Code length and code complexity have a negative correlation for the code quality produced by LLM [48–50].
- LLM occasionally does not provide user-requested code and might contain security and reliability hazards [48–50].
- Hallucinations observed in LLMs cause them to possibly find errors from the code that do not exist when asked to find them in feedback loops[48, 50].

The negative correlation between code length and complexity was found in studies [48–50]. In Liu *et al*. [48] code generation was tested in Python and Java programming tasks. The study found that the quality of LLM-generated code decreases with the generated code length and the required complexity of the code. In both Java and Python, the complexity and length of the code impacted the probability of correct results with a P value of < 0.01 [48]. In Dou *et al*. [49], LLM code generation was measured with different datasets taking into account their code length, code complexity, and number of API calls. Based on the comparison between datasets, the study found that mentioned code qualities lower the error-free code in both open and closer source LLM's [49]. In Chong *et al*. [50], which focused on the security abilities of LLM-generated code, buffer size makers in C were generated using LLM. The success rate was determined to be related to the complexity of the buffer, for example, a constant integer had a success rate of 99.7%, but adding it as a substation of a float lowered it to 50.1%. Therefore, the complexity of the code lowered the probabilities.

Another challenge of LLM is the production of safe and reliable code that does not meet user requirements [48–50]. In Liu *et al*. [48] only 4% of the errors were related to runtime errors, while badly styled and low maintainable code (47%) and wrong outputs (27%)

were the main categories of errors [48]. Badly styled and low maintainable code included categories like redundant modifiers, variables with ambiguous names, and too many local variables in a function/method. Each of the mentioned categories can be seen as a problem for safe and reliable code, as badly written code can contain risks for each requirement. In Dou *et al*. [49], the LLM functionality was identified as the most notable reason for errors. Especially understanding the provided problem from prompt and logical errors were prevalent [49]. Those errors included concepts such as adding corner checking and defining conditional branches for code.

In Chong *et al*. [50], 220 code files were compared between LLM and human-made versions. The study found that LLM generated less code to solve the problem but with a cost of security checks that exist in human-generated files. Additionally, the study tried to generate an SHA generation algorithm using LLM which failed, while algorithms AES and MD5 succeeded, making the algorithms unreliable from a security point of view [50].

The last finding was that LLM hallucinates non-existing errors in the code when asked to find errors from feedback loops [48, 50]. In Liu *et al*. [48], having a feedback loop to find errors in the code improved the code quality between 20-60% but added additional errors in the code. This happened especially if the feedback is generated with only LLM instead of using runtime and static feedback, which provided better results[48]. In Chong *et al*. [50], the feedback loop reduced the number of security problems like malloc overflow and null dereference by 40%. However, when security problems did not exist in a file, LLM wrongly added malloc errors while trying to fix a non-existing problem.

When found challenges are compared to the studied multi-agent systems, a potential improvement to them can be recognized. For feedback loops implemented in multi-agent systems [2, 43–45] corresponds to the benefits of [48, 50] despite the possibility of additional hallucination of errors. Especially using runtime errors as feedback as in Lei *et al*. [44, 45] or testing the code in Huang *et al*. [2] provides feedback with higher accuracy, which might be partial reasons for the high results in Table 2.2. With LLM generating code with low security and reliability, agents could be generated specifically tackling those matters. Nunez *et al*. [51] introduced a multi-agent system with an agent designed to find Common Weakness Enumeration (CWE) problems and was able to reduce the probability of CWE from 49% to 36% in a dataset used in GPT-4o.

For code length, a multi-agent system could be designed to make code in pieces. In Rasheed *et al*. [41], CodePori produces code in modules, allowing generated code to be generated in large amounts but in smaller generation sizes. For code complexity, multi-agent system could provide a specific agent to make a detailed plan. In Fourney *et al*. [1] an agent called as orchestrator is proposed, which creates a highly detailed plan of a given coding problem, which used in a multi-agent system produced good results in challenging benchmarks. The problems and solutions provided by the multi-agent systems

are collected in Table 2.3.

Table 2.3. *Multi-agent system solution to each found challenge in the literature*

Problem	Solution	References
Feedback loop hallucinations	Multi-agent system with runtime, test case, or static testing agent providing feedback	[2, 44, 45]
Low secure or reliable code generation	An agent in a multi-agent system that is designed to improve the security or reliability of the code	[51]
Code length and complexity	Make multi-agent system generate code in modules and include agents focusing on handling complexity	[1, 41]

2.4 Challenges in Upgrading Legacy Applications

A legacy system is a system that can be considered outdated. A system can be considered legacy due to the technology, programming language, hardware/software, or any other part being obsolete [52]. The strategies of handling a legacy system are disposing of the system, keeping it as it is, and re-engineer or replace components of the system [52]. This section discusses the challenges of upgrading legacy applications based on three case studies found in the literature and how LLM-based multi-agent systems could solve them. The cases focused on updating legacy applications and analysing challenges found during the process. The following challenges were found during the review:

- The programmers taking part in the updating process can have knowledge caps either in old or new technology related to application [53, 54]
- Understanding legacy applications input, output, and functionality of the code is a demanding, time taking task during updating process [53, 55]
- Breaking down the legacy updating process into tasks and managing them is a notable challenge during the project [53, 54]

The first challenge found was that programmers partaking in a legacy modernization project might have knowledge caps in the legacy or the modern technology related to the application [53, 54]. In De Marco *et al*. [53], a migration of a legacy mainframe application was committed to Linux servers. The study found that during feature development programmers knowledgeable in COBOL had challenges understanding Java and vice versa. This caused a recognized knowledge gap between the development teams [53]. In Fritzsch *et al*. [54], microservices migration in different phases was analysed. The study found that the shared first cause of recognized challenges was that there was not enough expertise in microservices with the developers. This was true in 8 of the 14 microservice migration cases studied [54].

The second challenge is the difficulty of understanding input, output, and functionality of the code of a legacy application [53, 55]. In De Marco *et al*. [53], the COBOL to

Java migration had a one-year delay during the project testing phase. The main cause of the delay was the lack of high-level test cases, which made the input and output requirements of different components a time-consuming task. Inside the components, the obsolete code also caused a delay as recognizing its functionality was difficult to correctly identify. In Vesić & Laković [55], a framework was proposed to evaluate a legacy system. It was used to analyse an existing legacy system collecting information for a drainage and sewer company. The analysis showed multiple different problems for the updating process evaluation. The system did not have proper documentation, there were not enough personnel, especially with a knowledge of the whole system, and the software architecture itself was defective [55]. This would cause problems with understanding the system's functions as in De Marco *et al*. [53], if the system were to be modernized.

The last challenge is the difficulty of breaking down the legacy updating process into smaller manageable tasks [53, 54]. In Fritzsch *et al*. [54] the second shared first challenge in updating microservices was the decomposition of an application into smaller parts. As with expertise, this challenge impacted 8 of 14 of the microservice migration projects studied [54]. In De Marco *et al*. [53] the project work packets were decomposed and used to forecast the duration of the project. However, the forecast failed notably, due to differences in batch orientation [53].

For these challenges, LLM-based multi-agent systems could offer solutions based on the findings in the literature, which are visible in Table 2.4. With the technology gap, LLM could provide assistance as LLM can be trained to have expertise in different technologies [28]. With a multi-agent system, an agent could be tasked to focus on a certain technology like Python as done with CodePori in Rasheed *et al*. [41]. By analysing the input/output of modules and their functionality, a system component dedicated to analysis could automate this recognition. Wen *et al*. [56] proposed Input/Output (I/O) specification called Gift4Code. Using a more detailed I/O specification, the pass rate of the coding metric improved during testing [56]. By managing the division of the legacy project into smaller tasks, LLM could automatize this project. In CodePori, for example, a manager agent divides coding tasks into smaller modules which are then executed in a pipeline, allowing the system to focus on each part separately [41].

Table 2.4. *LLM's solution to each found system modernisation challenge*

Problem	**Solution**	**References**
Knowledge gaps in technologies	LLM trained with all necessary technologies providing expertise as possible agents	[28, 41]
Understanding of legacy system functions	I/O analyser	[56]
Breaking down legacy project into modules	Having LLM breaking the project into smaller tasks from the project description/system analysis	[41]

3. RESEARCH METHODOLOGY AND MATERIAL

Chapter 3 provides the research methodology and the material of the study. The first section discusses the definition of how multi-agent systems and reasoning models improve accepted code probability mathematically. Then, the introduction of the proposed system follows and the reasoning behind its architecture. The case application and its files are presented next. The verification process for the proposed system follows this. Last, the validation process is explained with the GitHub files used and the related updating tasks done to them.

3.1 Probability Adjustment with Intermediate Tokens Between a Reasoning Model and a Multi-agent System

As found in Subsection 2.1.1, LLM returns the distribution of tokens as possible answers for each prompt. From that distribution, a certain percentage of possible answers will satisfy the task given by the user. As this thesis is limited to code-generating multi-agent systems, a definition for accepted code from answers is defined. In this thesis, the definition of the accepted code is as follows:

1. *Error-free:* The produced code generated by LLM contains no errors.
2. *Correct task:* The code can execute the defined task given by the user.

Based on the definition, the probability of an accepted code will be $p(\neg error \wedge task)$ from the distribution of Equation 2.1. In the definition $error$ means any possible error in the generated code and $task$, the task being completed successfully by the code.

For models and systems where a part of the output is not intended as a final answer for the user, this study uses the term intermediate tokens. In reasoning models, intermediate tokens are called reasoning tokens, which allow the model to refine and reflect its final answer for the purpose of a better answer [5]. This allows the generated tokens to be divided into two chains of tokens where intermediate tokens can be separated in Equation 2.1, giving the following equation:

$$p(output|input, task) = \prod_{i=1,j=i}^{n} p\left(y_j \mid in_1, \ldots, in_i, y_{i+1}, \ldots, y_{j-1}, input, task\right) \quad (3.1)$$

Where $in_1, \ldots, in_i$ are intermediate tokens and $y_{i+1}, \ldots, y_{j-1}$ are output tokens fulfilling the user's request.

With multi-agent systems, the intermediate token generation is different due to agents interacting with each other. Let us first define an LLM agent based on 2.1.

$$p(output|inputs, task) = \prod_{i=1}^{n} p\left(y_i|y_1, \ldots, y_{i-1}, inputs, tasks\right) \quad (3.2)$$

Where $inputs$ is a combination of the system prompt and any additional content from the previous agents. Therefore, intermediate tokens are not generated by the LLM query during execution, but they are generated beforehand by executing agents previously in the system. Equation 3.2 can be such formatted by separating agent-generated input tokens as intermediate tokens:

$$p(output|inputs, task) = \prod_{i=1,j=1}^{n} p\left(y_i|y_1, \ldots, y_{i-1}, in_1, \ldots, in_j, input, tasks\right) \quad (3.3)$$

Where $in_1, \ldots, in_j$ are intermediate tokens from the previous agents in the system and $input$ the system prompt given to the agent.

As each agent has a certain role, a single agent forms a persona pattern as defined in [42]. With multiple of these agents, it can be argued that a multi-agent system not only forms persona patterns but a team pattern where interactions of different persona patterns form a higher probability manipulation for the output. Intermediate tokens $in_1, \ldots, in_i$ are added by each agent interaction, as can be observed in Equation 3.2 by always producing a new output of intermediate tokens. As such, they hold the outputs of each agent, and the final output given to a user is therefore a result from a team pattern which is an output formed of an interaction of different agents.

Based on this establishment, multi-agent systems are advanced prompt engineering techniques that try to produce the correct output with the help of intermediate tokens. Unlike reasoning models, intermediate token generation is done by different agents that are given different roles in the system. The combination of different roles tries to give an output that is formed by a 'team' and is defined in this thesis as a team pattern. It is important to note that reasoning models and multi-agent systems are not exclusionary. An

agent can be reasoning-based in the system, it just means that there will be two layers of intermediate tokens, one for the agent's output and the output as intermediate tokens for the system as seen in equations 3.1 and 3.3.

3.2 Multi-agent System: Verifying Agent Pipeline Updater

The multi-agent system proposed in this thesis is named Verifying Agent Pipeline Updater (VAPU) to make a more clarifying name for the original multi-agent pipeline proposed in [13]. The system is referred from a multi-agent system CodePori, introduced in Rasheed *et al*. [41]. Codepori consists of six agents tasked to operate as a software development team to generate the desired applications for the client. The agents used in CodePori are a manager agent, two developer agents, two finalizer agents, and a verifier agent [41]. VAPU uses a manager, verifier, and finalizer agent in the proposed system with the addition of the developer agents changed into a prompt maker and an execution agent.

VAPU is designed to modify existing code based on both user-given requirements and the already existing codebase provided by the client. The requirements contain user specifications on how the code should be refactored. One possibility, for example, is to update the code from version X to version Y of the programming language or to make the code compatible with a different library. The system consists of four different units, which are explained below and illustrated in Figure 3.1.

Manager agent: The manager agent is meant to divide requirements into smaller manageable tasks, as done with CodePori. The manager agent receives the requirements from the client and rewrites them in an understandable format for LLM agents. The tasks are abstracted to be further detailed by following agents in the system. The manager agent is also asked to reflect on the decision and make sure that all tasks are in chronological order. The created tasks are then sent to the task pipeline for further processing.

Task pipeline: The task pipeline consists of two agent roles, a prompt maker agent and an execution agent, as visible in Figure 3.1. For every task received from the manager agent, a prompt maker agent creates a detailed prompt which is then used in the executing agent for the provided code. The code is then sent to the verification agent which returns after verification for the next task in the pipeline. After the pipeline has completed all the tasks, it sends the fully updated code back to the client.

Verification agent: The verification agent's task is to ensure that the task is completed by comparing old and new code and reflecting the result on the task and the project goal. By this determination, the agent's task is to make a list of found problems or announce that the task has been completed successfully. If the verifier agent accepts the task as completed, the code returns to the task pipeline as it is. If the agent finds problems in the code, it will send it to the finalizer agent for further adjustment.

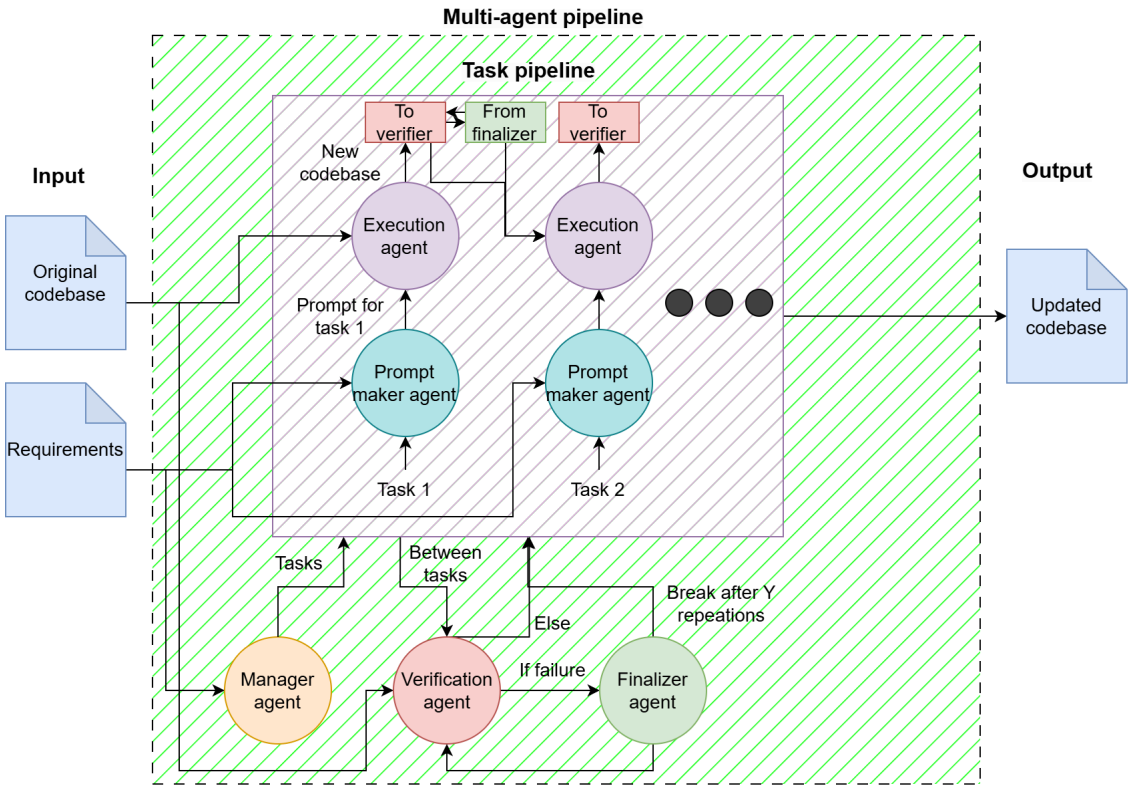


Figure 3.1. *Proposed system: Verifying agent pipeline updater VAPU reproduced from [13]*

Finalizer agent: The finalizer agent modifies the code based on the verifier agent's remarks on the provided code. After modification, the code is sent back to the verifier agent to check if the code is now fulfilling the task requested. If the task is still not complete, the finalizer agent makes a new version from the updated list of remarks given by the verifier agent. After a specific number of iterations in a feedback loop between the verifier and the finalizer agent, the finalizer agent will send the code to the task pipeline if the code does not converge for the task. This might happen due to hallucinations or if the task is too complex for the LLM used.

VAPU is designed to provide the following improvements for the recognized challenges found in Subsection 2.4:

1. *Self-division:* Liu *et al.* [36] observed the challenge of long prompts in LLMs where the information in the middle of a prompt is hard to access. Additionally, as found in Subsection 2.4, dividing legacy updating tasks into smaller parts is a recognized challenge. The manager agent of the system solves the problem of task management by dividing a project into smaller requirements. As such, a long prompt covering all the requirements is also not needed, which makes the information understanding more reliable.
2. *Self-feedback:* As mentioned in the agent's description, communication between

the verifier and the finalizer agent is iterative for code updating. As found in the background chapter, a self-feedback loop has the potential to improve code generation quality [48, 50]. As the feedback loop could refactor the code to fit fully to a task, a similar quality improvement is possible with the system.

3. *Self-instructing:* Writing complex prompts manually for LLM is a challenge, especially for those who are not trained in prompt engineering. In Zamfirescu-Pereira *et al.* [57], prompt writers without expertise were found to have generalizing and human interactive ways of making prompts. This resulted in bad performance when executed in LLM's [57]. Additionally, as found in Section 2.4, one legacy application challenge is unfamiliarity with technology. With self-instruction, the system can write information that could be left out by a client not familiar with the technology.

3.3 Compared LLM's

In this thesis, five different LLMs are used to study differences in performance using VAPU. The models were selected in different parameter ranges, with the assumption that the LLM quality for generating code would correspond to the parameter size by the scaling laws [4, 34]. Different models were queried using the OpenAI API and a routering website OpenRouter during the study [58, 59]. A total of five models were selected, including the one used in the original study [13]. Based on their estimated size, two of the models are labelled as large, two as medium size, and one as small. The selected models are shown in Table 3.1.

First in the table are the developer company and the model's name. As most companies do not reveal the parameter size in their LLM, the parameter estimations have been taken from Abacha *et al.* [60] for GPT-4o-mini, GPT-4o, and Claude 3.5 Sonnet. For DeepSeek-V3, the parameter size has been disclosed with 671B, parameters of which 37B are active [61]. As with Nova Pro 1.0, no parameter disclosure or estimation was found in the literature. Therefore, estimation was done based on the model's price in OpenRouter ($0.80/M input tokens, 3.20$/M output tokens), which is around fifth from GPT-4o's ($2.50/M input tokens, 10.00$/M output tokens), and Claude 3.5 Sonnet's price ($3.00/M input tokens, 15.00$/M output tokens) and was therefore estimated as 40 billion parameters in this thesis [59]. The context length has been taken from the OpenRouter website for each model [59]. Additionally, the hallucination rate of each model is visible in the table. Hallucination rates were taken from a hallucination evaluation model HHEM-2.1, which results are publicly available on GitHub by Vectara [62].

In Table 3.1, the size of the LLM is announced as 'L' for large, 'M' for medium, and 'S' for small LLM. Depending on the classification, GPT-4o-mini could be labelled an SLM in the literature [15, 16] and Nova Pro 1.0 with DeepSeek-V3 in some cases [16]. In this thesis, they are labelled as LLMs to keep the comparison simple. Labels are still given to

illustrate the differences in size for later discussion.

Table 3.1. *Selected LLMs for the thesis*

Company	Model	Parameter size [60, 61]	Context length [59]	Hallucination rate [62]	Size
Anthropic	Claude 3.5 Sonnet	~175B	200K	4.6%	L
Amazon	Nova Pro 1.0	~40B	300K	1.8%	M
DeepSeek	DeepSeek-V3	37B (671B)	128K	3.9%	M
OpenAI	GPT-4o-mini	~8B	128K	1.7%	S
OpenAI	GPT-4o	~200B	128K	1.5%	L

3.4 Case: Electronic Dictionary Project

The web application used as a case study in this thesis is called an Electronic Dictionary Project (EDP) based on the Dictionary of Medical Vocabulary in English introduced by Norri *et al.* [63]. The EDP is a management tool for a PostgreSQL database of medical vocabulary. It is used as a research assisting tool for different operations targeted for the database, which architecture is specified to contain the vocabulary and its metadata.

The EDP was originally created between 2007-2009 by three student teams [63]. In 2024, the application was revised with a new project from January to May. During this period, the application functionalities were extended and moved to the Docker environment. After the project, the remaining leftover from the previous version was the degraded web framework from 2008. In the risk assessment, this was considered to cause eventual problems in the running environment. Therefore, the need to update the application to a modern version of the web framework was deemed necessary.

The EDP was written in the 2008 version of CakePHP 1.2, which is a web framework built over the PHP programming language since 2005 [64]. In this thesis, the goal of answering **RQ1** and **RQ2** is updating selected files in the dictionary from CakePHP 1.2 to version 4.5 with any additional syntax and library changes required. CakePHP 1.2 was released in the year 2008 [65], and CakePHP 4.5 [66] on 2023 making the version jump required in total of 15 years.

The architecture of the EDP is shown in Figure 3.2. The current version runs inside a legacy PHP 5.6 image where related files are put under CakePHP framework application. EDP communicates with the dictionary's PostgreSQL database by loading and modifying the information inside it. Created as an MVC (Model-View-Controller) application, the view and model implementations are divided, with controller files acting as mediators between them. Additionally, the dictionary includes deprecated web application library YUI, webroot files, and JavaScript implementations.

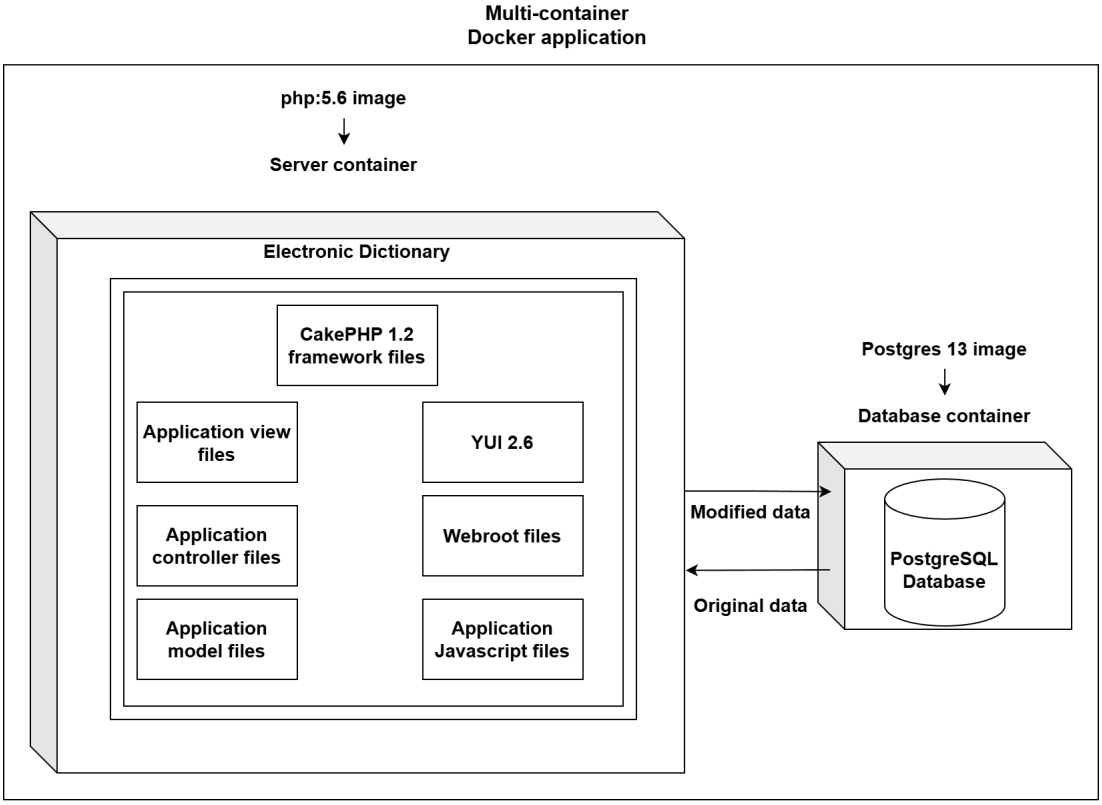


Figure 3.2. *Architecture of EDP*

The following qualities reason the use of this application in this thesis:

1. *Deprecated application:* An application built on a 15-year-old version of a web framework is suitable for testing VAPU as it can be considered as a legacy application.
2. *Detailed documentation of the web framework:* CakePHP is documented in high detail on the Internet with differences between versions explained, for example, on the web framework website [67]. Therefore, the LLM used is expected to have enough data to give noteworthy results in a wide range of tasks.
3. *Continued support of the web framework:* The web framework is actively updated, making the updated version verifiable in modern environments [67].

3.4.1 Used Files

The files used for the evaluation of VAPU were six view files from the application that belong to two different logical modules. The controller and model files were updated beforehand for version 4.5, leaving the view files to be updated. The view files are named from A to F, with LOCs varying between 19 and 190. The views, LOC, and the operations conducted on them are visible in Table 3.2.

Table 3.2. *Selected files from the EDP, adapted from [13]*

File	LOC	Operation
View A	35	Web framework update
View B	25	Web framework update, syntax change
View C	57	Web framework update, syntax change
View D	190	Web framework update
View E	19	Web framework update, form architecture change
View F	118	Web framework update, library change

View A is a list of dictionary terms that are related to a singular term. It is selected as the first test due to its low complexity and small LOC. View B is a list of quotes that contains a certain variant in the dictionary. View C is a search tool for the variants. Both View B and C require a syntax change due to the changed data format from the controller side as an Object-relational mapping (ORM) dataset, introduced in [68]. The difficulty is higher as the access to the dataset is done in an array style to the ORM object instead of a default PHP object operator.

View D is a dynamic dictionary list of all variants written with both PHP and JavaScript. With a much larger LOC than the rest of the files and two different coding languages, the view is challenging to update. Both View E and F are element template files used in View C. View E is an Ajax query for View C search which needs to be updated into JQuery search implementation. This means that the architecture in this view is changed with the addition of the CakePHP version update. View F is a highlighting element that is meant to highlight a variant in the returned list of quotes in View C. Additionally, View F contains four helper functions that enable the functionalities in the front end. During the update, the goal is to replace them with modern library implementations if possible.

3.5 Verification Process

The completion of the verification process for VAPU was a comparison to OSL and ZSL prompts depending on the scenario as in [13]. An OSL prompt was used in Views B and C, where the syntax change has been proven to give weaker results compared to a ZSL prompt [13]. The rest of the views were compared with the ZSL prompts. The use of ZSL and OSL prompts has been used as a metric for more complex prompt engineering methods, for example, in Ouédraogo *et al*. [69], prompts were used to compare against varying reasoning methods in LLMs.

Each view was updated ten times with each LLM to take into account the stochastic nature of the answers, as any used LLM did not produce a deterministic answer even with the temperature set to 0. If the updated file did not contain the whole code, the generation was considered as failed, and a new file was requested. In total, ten successful view instances were collected for each view and used LLM to answer **RQ1**. The update was carried out with the temperature parameter set to 1 and 0 during to answer **RQ2**. The maximum feedback loop of VAPU was set to two between the verification agent and the finalizer agent. Overall, 1200 file instances were created to conduct the verification process and to answer the RQs.

The process for Views A, B, C, and D was to count every occurred error once for the updated file. The count was done using both static and dynamic testing for the files. With static testing, the file was read, for example, to find any syntax errors. With dynamic testing, errors that occur in runtime where found. The found errors were divided into four main categories during the error recognition:

1. *Fatal Errors:* Errors that cause the studied view failing to run or cause a crash. This includes, for example, syntax errors or wrong library calls in the code.
2. *Runtime Errors:* Errors that allow running the view but cause some functionality to not work correctly. This can be an empty list where content should be visible.
3. *Content Errors:* The feature works, but its functionality differs from the original file.
4. *Missing/additional features:* The updated file is missing or has additional features or content that does not exist in the original view.

For Views E and F, a different method was used to evaluate the success of the update, as the views underwent full refactoring. The method chosen was to have different requirements expected from each updated view and to give a pass if that requirement functions correctly. A score of one was given for each correct function, and zero for failure. They were dynamically tested with different tests to check every functionality. The following requirements were set for the view files:

View E Requirements:

1. The form sends data to the controller and receives search results for the user.
2. The dropdown menu shows the correct suggestions for a searchable variant.
3. The search button and dropdown work when clicked without any errors occurring with the use.

View F Requirements:

1. The highlighting works in basic cases where wildcards or medieval English letters are not used.

2. Highlight works when medieval English letters are used in the variant.
3. Highlight works when wildcards are used. In this case, only the part until a wildcard is highlighted in the search result.

JQuery file necessary for View E was accepted in both URL and local file address. Also, the data name to fetch from the controller was changed due to a change in a controller function. With View F a guard was added to any remaining helper function to avoid re-declaration of functions during testing. No other changes were done to the views during the testing. With View F, the replaced functions were also counted.

The prompts created for each file are visible in Table 3.3. For views using ZSL prompts, the prompts were same with VAPU with an exemption of dividing each prompt into two requirements, with the first sentence of the prompt being the requirement one. With Views B and C the system prompt was different from the OSL prompt due to not needing an example. Unlike in [13], a ZSL prompt is not used in Views B and C, as the OSL prompt was shown to give better results and to make the comparison process same in every view file. The same applies for View F where the system was run with only two tasks. The last sentence for View B and C prompt were only used in View B as the requirement does not exist for View C.

For **RQ2** histograms were created for both temperatures to investigate the change in the distribution in the amount of errors and requirements for VAPU and OSL/ZSL method. Additionally, the ratios between temperatures were analysed to measure the relative change caused by the difference in temperature. In the verification process, the resulting LOC and the time taken to update the code were also measured. With LOC the average of the ten view file instances was investigated between VAPU and the corresponding OSL/ZSL prompt. With time, a scatter plot was done with normalized time to illustrate the number of errors observed in VAPU. The reason for this was to investigate, whether longer time spent in the multi-agent system lowers the rate of error with increased agent interaction and intermediate tokens as in Equation 3.3. Both code generation and system functionality in time were used to answer in **RQ3**.

3.6 Validation Process

Similarly to the verification process, the validation process was conducted by comparing VAPU to ZSL prompts with the same five LLMs used to update English dictionary files. The temperature used T^u in the verification process was the best performed value for each LLM/method based on the following formula: $T^u = AR - \frac{1}{2}AE$. AR is the average requirements, and AE is the average errors obtained during the process. As this is the validation process, each test was run as pass@1 to simulate realistic use of an application update and to answer the observed benefits of VAPU in **RQ4**. For the first part

Table 3.3. *Used prompts in each view, prompts first used in [13]*

File	Method	Prompt
View A	VAPU/ZSL	Update whole cakePHP view file from version 1.2 to version 4.5. Change $mainreference to $bookReference while removing ['Reference'] between wanted objects, access it ORM style. Write $term lowercase.
View B, C	OSL	Update CakePHP view file from version 1.2 to version 4.5. Requirement: $quotes are ORM \ResultSets made of arrays accessed with ['Fieldname']['fieldname'] and must be accessed so in the updated file. Example: old syntax: ['apple']['lemon'] new syntax: ['Apple']['lemon']. (Use function first() instead of [0] to the ORM object.)
View B, C	VAPU	Requirement1: Update whole CakePHP view file from version 1.2 to version 4.5. Requirement2: ORM Arrays must be accessed with array style syntax ['Fieldname']['fieldname'] with the first fieldname starting with a capitalized letter and the second only with lowercase letters. (Use first() when referring to first member in the array.)
View D	VAPU/ZSL	Update whole cakePHP view file from version 1.2 to version 4.5. Use ORM access with $variant with direct access to name and id.
View E	VAPU/ZSL	Update CakePHP version 1.2 ajax form into CakePHP 4.5 version with jQuery architecture including jquery-3.6.0.min file. Make the jQuery implementation fully functional version with dropdown updated every time the letter is written to it
View F	VAPU/ZSL	Update CakePHP element view file from 1.2 to 4.5. Replace functions in the element view file by ready made PHP libraries or update them if not found

of the validation process, VAPU was tested in 20 different GitHub Python repositories selected from open-source projects with varying backgrounds[70–78]. From each repository, a maximum of three projects were selected under the MIT license. The selected repositories and files are listed in Table 3.4 and Table 3.5. The tables list the used reposition, type of updated file, LOC, and cyclomatic complexity (CC), which is calculated using Radon, a Python metrics tool [79]. With Radon CC is announced from A to F depending on the complexity of the function, method, or class [79]. The CC is announced by the highest-ranking complexity in the file. The reasoning behind calculating LOC and CC is their usage to measure the difficulty of LLM generating code as found in Dou *et al.* [49]. Then, the tasks requested for the file are listed.

The selected files were chosen from the LOC range of 50-450, with the CC value between A-E, and each file, which were given from one to three different tasks. Each case was given an operation difficulty (OD), giving a point for every increase in a CC letter, a point for every task, and a point for every 100 LOC rounded to the nearest hundred. A maximum

Table 3.4. *Selected GitHub repositions and related information with OD 3-6*

Reposition/File	LOC	CC	Task(s)	OD
Python-Mini-Project [74] Simple_Http_Server /mhttp.py	64	A	1. Use http.server module [80]	3
python-beginner-projects [75] reciept generator/main.py	97	A	1. Add necessary receipt information [81]	3
The Algorithms - Python [71] other/password.py	97	B	1. Create a new function that products a strong password for it's checker	4
The Algorithms - Python [71] web_programming search_books_by_isbn.py	82	A	1. Fix for JSON keys handling 2. Console implementation to PyQt6 [82]	4
python-snake [72] snake.py	195	B	1. Console implementation to PyQt6 [82]	5
Amazing-Python-Scripts [76] Restaurant-Management-System /Restaurant-Management-System.py	226	A	1.Update time view 2. Add implementation for choice 'total'	5
Python-Mini-Projects [73] Alarm clock/alarm_clock.py	86	A	1. Reduce code using for loops 2. Tkinter to PySide6 [83] 3. Change drop-down selector to a number picker	5
Amazing-Python-Scripts [76] Health_Log_Book /main.py & database.py	259	A	1. Add calory estimation eat/spent 2. Matplotlib chart for calories [84]	6
Python-Mini-Projects [73] Bouncing_ball_simulator /ball_bounce.py	56	B	1. Add collision check between objects 2.Possibility to add/reduce objects 3. Add pause option	6
Python-Mini-Project [74] Finance_Tracker /main.py	161	A	1. tkinter to PySide6 [83] 2. Matplotlib to Plotly [85] 3.Possibility to remove income/expenses	6

of ten points were allowed for each file. The distribution of OD values was set between three and ten, with two to three files selected for each value. The reasoning behind this was to have a distribution of difficulty to evaluate how far each model and method is able to perform. The tasks range from library updates/changes and user interface changes to requirement fulfilment tasks. The prompts used in each method are listed in Appendix B.

Each task could get a maximum of three check marks on different tests. The first check mark was given after static testing to ensure that the code does not contain any of the old version code, and it contains the asked requirement change. The second check mark was given after code execution, ensuring it does not have any errors that would halt the execution and the main functionality works. The third check mark was granted if the functionality was as required and no runtime errors were encountered during the dynamic testing. This method combines the verification process steps by testing both errors and requirements, simulating a real use of the system in different coding tasks.

The benefits of VAPU were then evaluated by first comparing the check marks received against the compared ZSL prompts by the OD value. The OD value was then divided back to task amount, CC value, and LOC. From the division, the individual benefit in each category was calculated in a similar way by comparing the received check marks.

Table 3.5. *Selected GitHub repositions and related information with OD 7-10*

Reposition/File	LOC	CC	Task(s)	OD
Simple calculator [70] SimpleCalculatorPyQt1.py	342	B	1. PyQt from version 5 to 6 [82]	7
python-beginner-projects [75] Organize_Directory /organizer.py/main.py	90	D	1. Console implementation to PySide6 [83] 2.Option to select organized file types	7
Python-GUI-Project [78] Notepad /notepad.py	360	A	1. Tkinter to Kivy [86] 2. Add a possibility to convert text to PDF	7
Amazing-Python-Scripts [76] Address-Book /addressbook.py	259	B	1. Tkinter to PySide6 [83] 2.Fix database bugs 3. Add optional home address/email	8
The Algorithms - Python [71] geometry /geometry.py	288	B	1.Remove doctests 2.Add user interface with Kivy [86] 3.Add volume of a sphere	8
Python-Projects [77] Sudoku-Solver /SudokuGUI.py	316	E	1. Add pyttsx3 for audio with used commands [87]	9
python-beginner-projects [75] Scientific-Calculator /Scientific_CalC.py	378	C	1. Tkinter to PySide6 [83] 2. Add error handling for calculators (at least 3 different errors)	9
Python-Mini-Project [74] Lazy_Pong /pong.py	436	C	1. Move parser arguments to user interface 2. Allow paddle move to left and right	9
Python-Mini-Projects [73] Billing_system/Billing_system.py	428	D	1.Search option for name and phone number 2.Edit existing bills	10
Python-GUI-Project [78] cafe management system /cafe-management.py	397	C	1. Tkinter to PyQt6 [82] 2. Ability to change prices in an user interface 3. Discount option	10

4. RESULTS

This chapter presents the results of the thesis. The first section shows the error rates and the completed requirements of the updated view files A-E between an OSL/ZSL prompt and VAPU in different LLMs. The second section presents the differences in performance between the compared temperatures. Then, the following section presents the results of the validation process with the updated GitHub files. The last section presents the types of errors categorized and the results of the VAPU's time analysis. The raw results of the updating of the view files are available in Appendix A A.

4.1 Performance of VAPU in Different LLMs (RQ1)

Views A-F were updated according to the verification process explained in section 3.5. The results of views A-D are presented in Table 4.1 with temperature set to 1 and Table 4.2 with temperature set to 0. The results of views E-F are presented in Table 4.3, with the temperature set to 1, and in Table 4.4 with the zero temperature. The results for each view are displayed as an average of the ten runs with each run and summed together as a total value. Standard deviations are calculated from the sum of each view file in one run, which accounts for a total of ten instances for each model/method illustrated in the following equation 4.1:

$$\sigma = \sqrt{\mathbb{E}\left[(X-\mu)^2\right]},$$
$$X = err(ViewA) + err(ViewB) + err(ViewC) + err(ViewD) \quad (4.1)$$

The results obtained in GPT-4o-mini in views A-D and F when the temperature was set to 1 were first presented in Ala-Salmi *et al*. [13]. View E updating was completed wholly in this thesis due to providing the updated controller in contrast to the previous study. With Claude 3.5 Sonnet, there was also a need to modify the prompt with View D when the temperature was set to 0 due to a faulty returning format with a text: "Only view side is returned as it was given." which was added in the prompt.

Table 4.1 shows that VAPU's solutions produce more errors than OSL/ZSL prompts in

each model. The error difference was highest with GPT-4o-mini and Claude 3.5 Sonnet with 1.0 errors and lowest with DeepSeek-V3 with 0.60 errors for the benefit of OSL/ZSL prompts. In total, the average produced errors with all models was 2.02 with OSL/ZSL and 3.30 with VAPU. This makes the system produce, on average, around one error more for each update. Overall, DeepSeek-V3 produced the fewest errors in both methods, while Nova Pro 1.0 produced the most errors in OSL/ZSL, with an average of 3.00 errors. GPT-4o-mini had the highest error count in VAPU with an average of 4.40 errors.

Table 4.1. *Average different errors by model and method in a View files A-D with temperature set to 1*

Model	Method	View A	View B	View C	View D	Total	SD
GPT-4o-mini	OSL/ZSL	0.00	0.40	0.70	1.40	2.50	1.285
GPT-4o	OSL/ZSL	0.40	0.00	0.30	1.40	2.10	0.943
DeepSeek-V3	OSL/ZSL	0.00	0.10	1.00	0.00	1.10	0.300
Claude 3.5 Sonnet	OSL/ZSL	0.40	0.00	0.00	1.00	1.40	0.663
Nova Pro 1.0	OSL/ZSL	0.10	1.00	1.10	0.80	3.00	0.894
Average	OSL/ZSL	0.18	0.30	0.62	0.92	2.02	0.817
GPT-4o-mini	VAPU	0.50	0.60	1.00	2.30	4.40	1.200
GPT-4o	VAPU	0.50	0.20	0.30	1.60	2.70	1.345
DeepSeek-V3	VAPU	0.00	0.10	0.50	1.10	1.70	1.100
Claude 3.5 Sonnet	VAPU	1.20	0.20	0.40	1.50	3.30	0.943
Nova Pro 1.0	VAPU	0.80	1.10	1.40	1.10	4.40	1.855
Average	VAPU	0.60	0.44	0.72	1.52	3.30	1.289

When compared to the individual view files, View A was the most successful for OSL/ZSL prompts, while View B was the best for VAPU. In both cases, the views had the least errors in three different models. View D was the most difficult for both methods, being the only file where average errors did rise above 2 with the most models and methods. View C was also challenging to update, with an error rate higher in three cases compared to View D.

The standard deviation was usually higher with VAPU, with an exception with GPT-4o-mini with a lower value. The mean standard deviation of all models was 0.817 with OSL/ZSL and 1.289 with the system. From the models, DeepSeek-V3 did have the lowest deviation of 0.300 in OSL/ZSL and Claude 3.5 Sonnet with a value of 0.943 using VAPU. The highest deviation was with GPT-4o-mini in OSL/ZSL with 1.285, and with VAPU, it was 1.857 using Nova Pro 1.0.

From Table 4.2, VAPU gave better results in two models: GPT-4o-mini and DeepSeek-V3, while tracking the three other OSL/ZSL implementations. The average error rate of OSL/ZSL increased from 2.02 to 2.04 and with VAPU decreased from 3.22 to 2.30, making them closely equal in performance. The highest difference between the methods

was with Nova Pro 1.0, where the system scored 1.90 more errors against the OSL/ZSL implementation, which resulted in 0.40 errors on average. With DeepSeek-V3 VAPU also had 0.40 errors in total, making those two instances the shared best outcome of all 20 test scenarios. At the same time, GPT-4o-mini produced the worst score of 5.40 using OSL/ZSL.

Table 4.2. *Average different errors by model and method in a View files A-D with temperature set to 0*

Model	Method	View A	View B	View C	View D	Total	SD
GPT-4o-mini	OSL/ZSL	0.00	0.90	3.00	1.50	5.40	0.663
GPT-4o	OSL/ZSL	0.00	0.00	0.00	2.00	2.00	0.000
DeepSeek-V3	OSL/ZSL	0.00	0.00	1.00	0.00	1.00	0.000
Claude 3.5 Sonnet	OSL/ZSL	0.40	0.00	0.00	1.00	1.40	0.490
Nova Pro 1.0	OSL/ZSL	0.00	0.20	0.20	0.00	0.40	0.490
Average	OSL/ZSL	0.08	0.22	0.84	0.90	2.04	0.329
GPT-4o-mini	VAPU	0.20	0.30	1.30	1.70	3.50	1.565
GPT-4o	VAPU	0.60	0.00	0.60	1.90	3.10	1.446
DeepSeek-V3	VAPU	0.00	0.10	0.10	0.20	0.40	0.490
Claude 3.5 Sonnet	VAPU	1.00	0.20	0.00	1.00	2.20	0.872
Nova Pro 1.0	VAPU	0.10	0.50	0.60	1.10	2.30	1.005
Average	VAPU	0.38	0.22	0.52	1.18	2.30	1.076

In individual views a decrease of errors is mostly seen, for example, with View A five out of ten scenarios produced zero errors making it total of 50 error free files compared to three with a higher temperature. However, some cases had an increase like View C with GPT-4o mini where error rate rose from 0.70 to 3.00 using OSL/ZSL. As with higher temperature

The standard deviation decreased with the lower temperature with no variation in errors with GPT-4o and DeepSeek-V3 using OSL/ZSL. However, with VAPU, the deviation stayed high and even grew with GPT-4o from 1.345 to 1.446 and with GPT-4o-mini from 1.200 to 1.565. GPT-4o-mini additionally had the highest deviation in both methods and overall DeepSeek-V3 had in total the lowest deviation when both methods were compared.

Views E and F were updated with the methods explained in Section 3.5. As no OSL prompts were used in these views the method is listed only as ZSL. Similarly to views A-D the tables list the average of completed requirements in both files and the sum of them as total. The standard deviation is calculated similar to views A-D from the sum of average completed requirements from Views E and F. Additionally, the average of replaced functions from View F is counted if they are fully replaced by library functions in CakePHP.

From Table 4.3 the best performing model and method was GPT-4o with the average requirements of 3.20 out of 6.00. With VAPU the best performance was with DeepSeek-V3 with a score of 2.10 out of 6.00. The lowest performance in ZSL was on the contrast DeepSeek-V3 with an average of 0.90 and with VAPU Claude 3.5 Sonnet with an average of 0.60. The average requirements with ZSL was 2.08 and with VAPU 1.38.

In individual files both View E and F performed better with ZSL with an average value of 0.24 with View E and 0.44 with View F. The replaced functions with View F were notably higher with ZSL with 3.36 out of 4.00 compared to 1.74 observed in VAPU. The standard deviation of the accepted requirements was around equal between the methods with 1.158 with ZSL and 1.253 on average.

Table 4.3. *Average accepted requirements of View E and F and the replaced functions of View F with the temperature set to 1*

Model	Method	View E	View F	Total	SD	RF
GPT-4o-mini	ZSL	0.50	1.80	2.30	1.418	3.20
GPT-4o	ZSL	1.10	2.10	3.20	1.536	3.10
DeepSeek-V3	ZSL	0.90	0.00	0.90	0.300	3.80
Claude 3.5 Sonnet	ZSL	1.10	1.60	2.70	1.269	3.80
Nova Pro 1.0	ZSL	0.80	0.50	1.30	1.269	2.90
Average	ZSL	0.88	1.20	2.08	1.158	3.36
GPT-4o-mini	VAPU	0.70	0.50	1.20	1.166	2.70
GPT-4o	VAPU	1.00	0.60	1.60	1.429	1.40
DeepSeek-V3	VAPU	1.00	1.10	2.10	1.446	1.30
Claude 3.5 Sonnet	VAPU	0.00	0.60	0.60	1.200	0.60
Nova Pro 1.0	VAPU	0.50	0.90	1.40	1.200	2.70
Average	VAPU	0.64	0.74	1.38	1.278	1.74

From Table 4.4 the best result was obtained with Claude 3.5 Sonnet using OSL/ZSL, with total score of 5.20 from 6.00. The model had a perfect record with View E update, and was the only case where all ten outcomes were fully correct. With the second place was GPT-4o, with VAPU reaching an average score of 4.80 out of 6.00. This time, the lowest performance was shared between three models in ZSL GPT-4o-mini, DeepSeek-V3, and Nova Pro 1.0 with an average value of 1.00. With VAPU, Claude 3.5 Sonnet resulted in the same lowest value of 1.00. The average accepted requirements with ZSL was 2.50 and with VAPU 2.46.

In these scenarios, VAPU performed better with View F with a 0.36 higher average value, and ZSL retained the highest performance in View E with a 0.40 higher average value. The average replaced functions of View F were 3.10 with ZSL and 1.96 with VAPU. Similar to error rate, the standard deviation dropped with ZSL from 1.158 to 0.281 while with VAPU the change was low from 1.278 to 1.143.

Table 4.4. *Average accepted requirements of View E and F and the replaced functions of View F with the temperature set to 0*

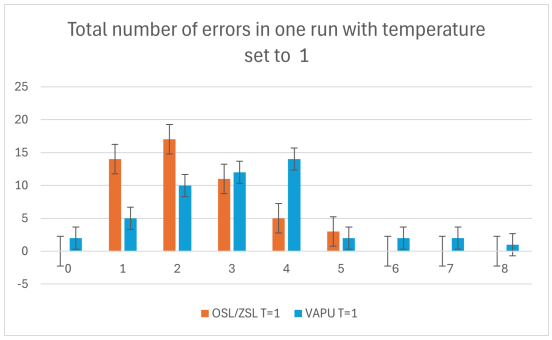
Model	Method	View E	View F	Total	SD	RF
GPT-4o-mini	ZSL	1.00	0.00	1.00	0.000	4.00
GPT-4o	ZSL	1.60	2.70	4.30	1.005	1.30
DeepSeek-V3	ZSL	1.00	0.00	1.00	0.000	3.30
Claude 3.5 Sonnet	ZSL	2.20	3.00	5.20	0.400	4.00
Nova Pro 1.0	ZSL	1.00	0.00	1.00	0.000	2.90
Average	ZSL	1.36	1.14	2.50	0.281	3.10
GPT-4o-mini	VAPU	0.80	1.40	2.20	1.661	0.50
GPT-4o	VAPU	1.90	2.90	4.80	0.600	1.60
DeepSeek-V3	VAPU	1.10	1.30	2.40	1.281	2.90
Claude 3.5 Sonnet	VAPU	0.10	0.90	1.00	1.130	1.60
Nova Pro 1.0	VAPU	0.90	1.00	1.90	1.044	3.20
Average	VAPU	0.96	1.50	2.46	1.143	1.96

4.2 Performance of VAPU in Different Temperatures (RQ2)

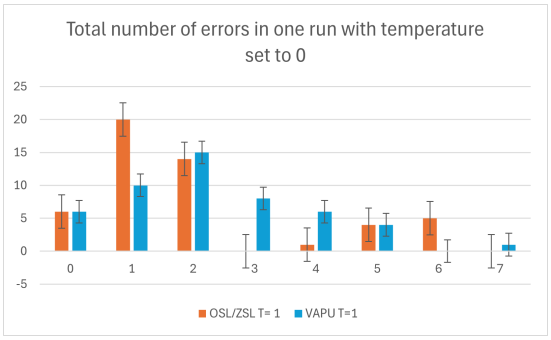
This section presents the detected differences between the two compared temperature values. First, histograms were made for both temperatures, which display the number of errors in each run with all LLMs combined with both methods visible. Both histograms are visible in Figure 4.1 for errors and Figure 4.2 for requirements.

In Figure 4.1, the distribution of errors in VAPU follows visually close to the Gaussian distribution at both temperatures. While with OSL/ZSL, the results converge with low and high errors at 0 temperature while maintaining similarity to the Gaussian distribution with temperature set to 1. The range of errors is higher with the system ranging from 0 to 8 and OSL/ZSL from 1 to 5 with the temperature set to 1. With the temperature set at 0, OSL/ZSL had an error amount between 0 to 2 and 4 to 6, and with VAPU 0 to 5 and 7. The temperature change improved the system performance by changing the distribution mode from 3 and 4 to 2 and adding the probability of lower error outcomes. With the OSL/ZSL, this transition did occur less notably, but the mode did change from 2 to 1. The probability of zero errors did increase to the same amount between methods, being 6 out of 50 cases, meaning a 12% chance of a complete successful update with zero temperature. With the temperature set at 1, the change was zero with OSL/ZSL and only 4% using the system.

In Figure 4.2, with the temperature set at 1, both methods had a high concentration at the low end of the completed requirements, as seen in the histogram. The outcomes with three or fewer requirements were 41 in ZSL and 46 with VAPU with the temperature set to 1. When the temperature was set to 0, those results were 31 with ZSL and 35



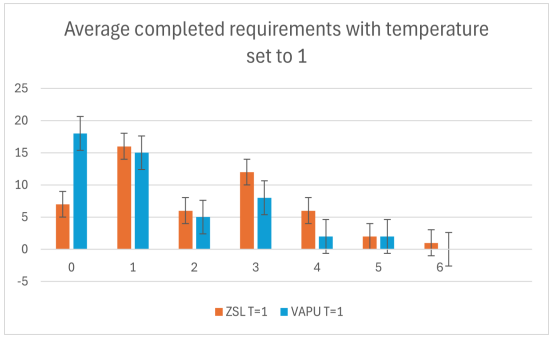
(a) The total number of errors in each iteration when $T = 1$



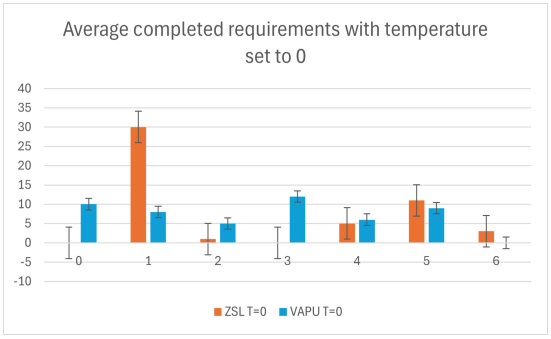
(b) The total number of errors in each iteration when $T = 0$

Figure 4.1. *Number of errors between OSL/ZSL and VAPU on each run*

with VAPU. Additionally, ZSL started to concentrate on only one requirement value, while VAPU had quite an equal distribution from 0 to 5 requirements, highlighting the highly varying results between the models using VAPU. The mode remained with ZSL as 1 with a stronger frequency while with VAPU it did increase from 0 to 3. The completion rate of all requirements increased with the reduction in temperature from 1 to 3 cases with ZSL, suggesting a 5% probability for full success. With VAPU, no case was fully accepted despite having a higher mode with the lower temperature.



(a) Completed requirements in each iteration when $T = 1$



(b) Completed requirements in each iteration when $T = 0$

Figure 4.2. *Completed requirements between OSL/ZSL and VAPU on each iteration*

The ratio of errors between different temperatures is visible in Table 4.5 with both methods and all used LLMs. The average error amount with OSL/ZSL was 3% higher on average with all models. With VAPU, the amount was 33% lower on average with the lower temperature. The ratio differences between models are high, with results differing notably in two models with OSL/ZSL with one improving and one worsening. With VAPU four models improved. As such, the temperature seems to either worsen, stay the same, or improve the average errors. However, with OSL/ZSL, the errors stayed in the majority of models nearly unchanged, and with VAPU the result was more likely to improve, as seen in the average ratio value.

Table 4.5. *Ratio of average errors by temperature*

	GPT-4o mini	GPT-4o	DeepSeek -V3	Claude 3.5 Sonnet	Nova Pro 1.0	Average
OSL/ZSL T = 0	5.40	2.00	1.00	1.40	0.40	2.04
OSL/ZSL T = 1	2.50	2.10	1.10	1.40	3.00	2.02
Ratio	2.160	0.952	0.909	1.000	0.1333	1.031
VAPU T = 0	3.50	3.10	0.40	2.20	2.30	2.30
VAPU T = 1	4.40	2.70	1.70	3.30	4.40	3.30
Ratio	0.800	1.150	0.240	0.670	0.520	0.670

The ratio of average requirements had a higher improvement in both models when the temperature was lowered, as visible in Table 4.6. With ZSL prompts, the improvement was 11.7 % on average, with two models notably improving and two worsening. With VAPU, the average improvement was 80.0% on average, with all models improving. The rate of improvement varied highly between models, with VAPU the LLMs improved between 14.3%-200.0%. Similar to the error ratios, ZSL implementations could either improve, worsen or stay around the same, while with VAPU there was only improvement with the temperature lowering.

Table 4.6. *Ratio of average requirements by temperature*

	GPT-4o mini	GPT-4o	DeepSeek -V3	Claude 3.5 Sonnet	Nova Pro 1.0	Average
ZSL T = 0	1.00	4.30	1.00	5.20	1.00	2.46
ZSL T = 1	2.30	3.20	0.90	2.70	1.30	2.08
Ratio	0.435	1.345	1.111	1.926	0.770	1.117
VAPU T = 0	2.20	4.80	2.40	1.00	1.90	2.44
VAPU T = 1	1.20	1.60	2.10	0.60	1.40	1.34
Ratio	1.833	3.000	1.143	1.667	1.357	1.800

4.3 Challenges of a Multi-agent System (RQ3)

As explained in the verification process, the types of errors were collected and are visible in Table 4.7 for the temperature set to 1 and Table 4.8 for the temperature set to 0. The tables list the used model and the method. This is followed by the number of the four different error types and their total number.

From Table 4.7, fatal and runtime errors were the most occurring errors found. The total errors was 101 with OSL/ZSL prompts and 165 with VAPU. With OSL/ZSL prompts 18.8% were fatal, 78.2% runtime, and 3.0% content errors. With VAPU 32.1% were fatal, 60.0% runtime, 6.7% content, and 1.2% missing content errors. In Table 4.8 the runtime

and fatal errors were also mostly occurring errors. The total error amount was 102 in OSL/ZSL and 115 with VAPU. The error rate was 35.3% of fatal and 64.7% of runtime errors with OSL/ZSL. With VAPU the ratio was 30.4% with runtime, 67.0% with fatal, 1.7% with content, and 0.9% with missing content errors.

Table 4.7. *Error types by each model and methods with temperature set to 1*

Model	Method	Fatal	Runtime	Content	Missing content	Total
GPT-4o-mini	OSL/ZSL	9	16	0	0	25
GPT-4o	OSL/ZSL	4	14	3	0	21
DeepSeek-V3	OSL/ZSL	0	11	0	0	11
Claude 3.5 Sonnet	OSL/ZSL	5	9	0	0	14
Nova Pro 1.0	OSL/ZSL	1	29	0	0	30
Total	OSL/ZSL	19	79	3	0	101
GPT-4o-mini	VAPU	10	34	0	0	44
GPT-4o	VAPU	9	18	0	0	27
DeepSeek-V3	VAPU	0	13	3	1	17
Claude 3.5 Sonnet	VAPU	19	11	3	0	33
Nova Pro 1.0	VAPU	15	23	5	1	44
Total	VAPU	53	99	11	2	165

Runtime errors in both tables consisted mostly of different errors in hyperlinks and wrong access to ORM-arrays, which were mentioned in the given prompts. The runtime errors did occur in every test scenario and was sometimes lowered with temperature but could also get higher. Overall, the differences between OSL/ZSL and VAPU were not much different with runtime errors. However, between scenarios, Nova Pro 1.0 with OSL/ZSL and zero temperature had only 2 errors and DeepSeek-V3 with zero temperature VAPU had 3 errors which in contrast to some scenarios measuring over 30 errors indicates high variance between a model and method.

Fatal errors mostly consisted with hallucinations like using a non existing cakePHP helper, using wrong operations to objects or keeping the deprecated CakePHP 1.2 code instances as a part of a view. The last mentioned was the reason for the high increase of fatal errors in GPT-4o-mini using OSL/ZSL. Noticeably, DeepSeek-V3 had not a single fatal error in any testing case, which is an interesting outcome as one of the smaller models. The content and missing content errors were uncommon and focused mostly with VAPU, including, for example, an unnecessary repetition of a view component.

As mentioned in section 3.5, both the used time and LOC were measured and added to the results which are fully visible in Appendix A. The code length of each file was measured and ALOC was calculated from each ten files generated by method and temperature. The obtained ALOC values are visible in Table 4.9 for temperature set at 1 and

Table 4.8. *Error types by each model and methods with temperature set to 0*

Model	Method	Fatal	Runtime	Content	Missing content	Total
GPT-4o-mini	OSL/ZSL	30	24	0	0	54
GPT-4o	OSL/ZSL	0	20	0	0	20
DeepSeek-V3	OSL/ZSL	0	10	0	0	10
Claude 3.5 Sonnet	OSL/ZSL	4	10	0	0	14
Nova Pro 1.0	OSL/ZSL	2	2	0	0	4
Total	OSL/ZSL	36	66	0	0	102
GPT-4o-mini	VAPU	4	31	0	0	35
GPT-4o	VAPU	6	25	0	0	31
DeepSeek-V3	VAPU	0	3	0	1	4
Claude 3.5 Sonnet	VAPU	16	5	1	0	22
Nova Pro 1.0	VAPU	9	13	1	0	23
Total	VAPU	35	77	2	1	115

Table 4.10 for temperature set at 0 with the model, method, and temperature used followed by LOC of view files A-F. Additionally, the original values are highlighted in the first row.

Table 4.9. *ALOC of each View file by method with temperature set to 1*

Model	Method	View A	View B	View C	View D	View E	View F
Original	-	35	25	57	190	19	118
GPT-4o-mini	OSL/ZSL	35	22	57	160	44	42
GPT-4o	OSL/ZSL	34	24	61	155	42	49
DeepSeek-V3	OSL/ZSL	35	23	56	170	54	32
Claude 3.5 Sonnet	OSL/ZSL	40	26	69	170	65	34
Nova Pro 1.0	OSL/ZSL	35	24	60	161	48	49
GPT-4o-mini	VAPU	36	24	60	166	66	52
GPT-4o	VAPU	35	32	63	163	42	73
DeepSeek-V3	VAPU	35	25	59	171	61	82
Claude 3.5 Sonnet	VAPU	47	39	75	166	150	113
Nova Pro 1.0	VAPU	41	25	66	168	45	59

Starting with View files A-D in both tables, the difference in generated code is not much, as the code was only updated to the new version of CakePHP with syntax changes. The most notable scenarios from the results in views A-D is Claude 3.5 Sonnet in both tables that resulted in more code, especially in views A and B, which is mostly explained by the use of dog strings. Compared to the original code, View D was reduced with an average of 9 LOC in Table 4.9 and 19 LOC in Table 4.10.

With views E-F there is a lot of variation between models and methods in both tables.

VAPU did generate more code with View E ranging from 42-150 LOC and View F between 52-113 LOC as opposed to 42-65 LOC with View E and 32-49 LOC with View F using OSL/ZSL in Table 4.9. With Table 4.10 this range was 46-141 with View E and 58-87 with View F using VAPU and 37-70 with View E and 32-72 with View F. Notably, code generated with Claude 3.5 Sonnet using VAPU was much higher compared to any other case with a 2-4 times higher LOC.

Table 4.10. *ALOC of each View file by method with temperature set to 0*

Model	Method	View A	View B	View C	View D	View E	View F
Original	-	35	25	57	190	19	118
GPT-4o-mini	OSL/ZSL	35	23	56	165	41	32
GPT-4o	OSL/ZSL	35	23	57	167	37	72
DeepSeek-V3	OSL/ZSL	35	23	56	161	60	40
Claude 3.5 Sonnet	OSL/ZSL	41	23	78	182	70	32
Nova Pro 1.0	OSL/ZSL	35	24	57	183	42	52
GPT-4o-mini	VAPU	35	24	60	170	53	68
GPT-4o	VAPU	35	28	66	172	46	74
DeepSeek-V3	VAPU	35	23	61	159	63	58
Claude 3.5 Sonnet	VAPU	44	37	76	183	141	87
Nova Pro 1.0	VAPU	38	25	66	168	48	47

Ovearall, VAPU did generate more code, but in most cases the difference is only a few LOC as seen from the tables. With the temperature comparison between each view produced by a model and method from the tables, LOC was 23 times higher, 13 times equal, and 24 times lower with the temperature set to 1. This indicates that the temperature does not have any impact on either direction with the amount of LOC.

To analyse impact of time for models performance with different view files, the durations obtained from each run were normalized using the following equation 4.2:

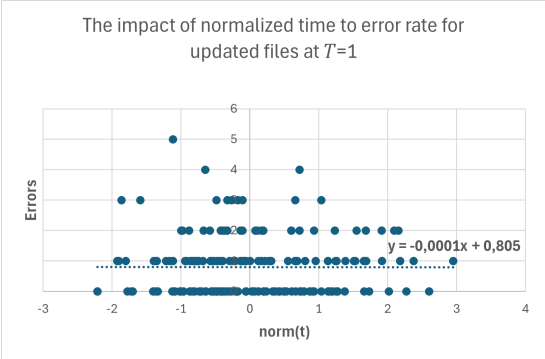
$$\mathrm{norm}(t) \;=\; \frac{x-\mu}{\sigma} \tag{4.2}$$

Where mean and standard deviation are calculated from the ten runs of each view file by method, model, and temperature. As such, the results will show if the system spending more time for the updating process has an impact on its performance. The results of the normalization and its relationship with VAPUs performance using normalization are visible in Figure 4.3. To show the trend of times impact linear regression was calculated to show whether time has a positive or negative impact to the systems performance.

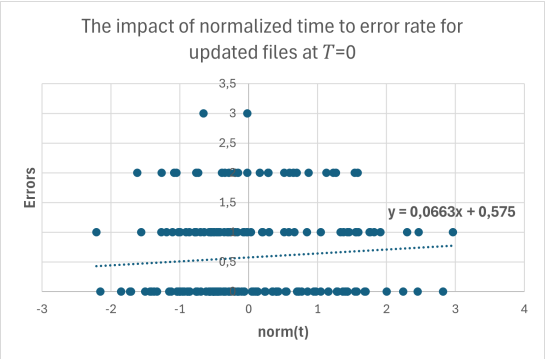
Starting with error rate in temperature set to 1, no visible change happened to the performance and linear regression shows nonexistent change for error rate with -0.00001 errors

by increase of each standard deviation. With temperature set to 0, the baseline error rate for each view file drops from 0.805 to 0.575 but the rate of errors increased with time slightly by 0.0663 errors with each standard deviation. Time also had a slightly negative impact to fulfilled requirements also as -0.0488 requirements were lost with the temperature set to 1 with each standard deviation. With the temperature set to 0 this impact was slightly lesser but still negative with a value of -0.0297. The average requirements fulfilled by each view increased from 0.6889 to 1.0125.

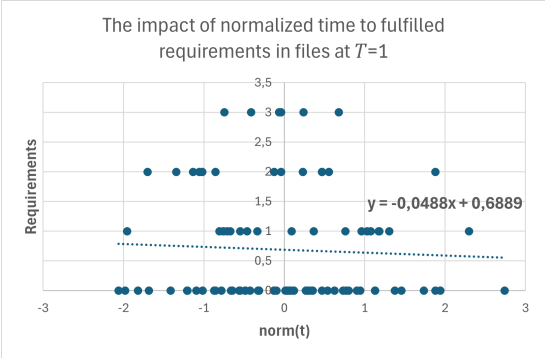
Based on these findings, time has a slight negative impact on system performance in most cases. The results demonstrate that additional verification runs or longer token generation by the agents do not start converge the system towards the wanted outcome, but keep it mostly unchanged.



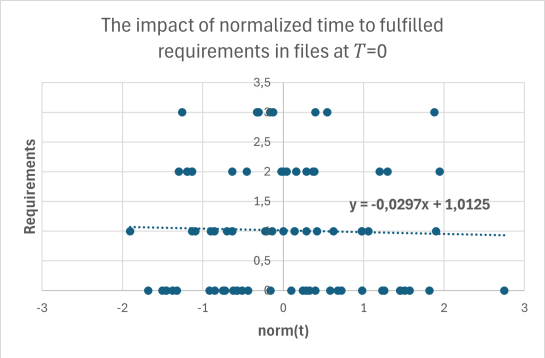
(a) Error rate at $T = 1$



(b) Error rate at $T = 0$



(c) Fulfilled requirements at $T = 1$



(d) Fulfilled requirements at $T = 0$

Figure 4.3. *Scatter plot of errors and requirements using VAPU with normalized time*

4.4 Benefits of a Multi-agent System (RQ4)

As stated in the validation process, the temperature used was selected based on the equation $T^u = AR - \frac{1}{2}AE$, these values are calculated in Table 4.11 based on the results obtained in Sections 4.1 and 4.2. As seen in Table 4.11, nine out of ten tests were performed with the temperature set at 0 with the exception of GPT-4o mini, which had a notable downgrade with OSL/ZSL prompts with the temperature set at 0.

Table 4.11. *Used temperature of each method in the verification process*

Model	**Method**	$T = 1$	$T = 0$	T^u
GPT-4o-mini	OSL/ZSL	1.05	-1.70	1
GPT-4o	OSL/ZSL	2.20	2.95	0
DeepSeek-V3	OSL/ZSL	0.35	0.50	0
Claude 3.5 Sonnet	OSL/ZSL	2.00	4.50	0
Nova Pro 1.0	OSL/ZSL	-0.20	0,60	0
GPT-4o-mini	VAPU	-1.00	0.45	0
GPT-4o	VAPU	0.05	3.25	0
DeepSeek-V3	VAPU	1.25	2.20	0
Claude 3.5 Sonnet	VAPU	-1.05	-0.10	0
Nova Pro 1.0	VAPU	-0.80	0.75	0

The results of 20 Python GitHub tasks are visible for ZSL prompts in Table 4.12 and for VAPU in Table 4.13, which are ordered by their OD value. Due to the length of the longest Python files, some non-critical code had to be reduced with longest code files due to output length being limited or the model refusing to produce long token outputs. These are marked with [1]. In total, the ZSL methods obtained 189 check marks and VAPU 205 check marks across all Python tasks and LLMs. The average amount of check marks was 38 in ZSL and 41 in VAPU between the different LLMs.

The best performing LLM was Claude 3.5 Sonnet with a total of 46 check marks with ZSL while the second best was GPT-4o with 45 check marks with VAPU. The lowest score performed was with Nova Pro 1.0 with ZSL, while the lowest score in VAPU was shared between three models GPT-4o mini, DeepSeek-V3, and Nova Pro 1.0. This also indicates that the performance of the lowest model increased by 22.5% with the higher check mark amount observed.

With the individual LLMs VAPU had one check mark more with GPT-4o-mini and DeepSeek-V3. ZSL had one check mark more with Claude 3.5 sonnet. Both GPT-4o and Nova Pro 1.0 were seven check marks higher with VAPU. Therefore, two LLMs had a notable increase in check marks with VAPU, for Nova Pro 1.0 the before mentioned increase of 22.5% and 18.4% for GPT-4o. Taking into account all LLMs, the expected improvement is 8.5%.

Regarding OD, it can be seen visually that the number of succeeded file updates decreases in the highest values in both tables. This is illustrated in Table 4.14, where the total check marks of all LLMs used are set in increments of 3-4, 5-6, 7-8, and 9-10 for both ZSL prompts and VAPU. Both methods had the highest average value with OD value between 3-4 with ZSL 11.00 and VAPU with 12.25. With higher values of 5-6 and 7-8 the average stays around the same, with both methods having 10.16/10.46 with ZSL and

Table 4.12. *Results of GitHub updates with ZSL by difficulty*

Project	GPT-4o mini	GPT-4o	DeepSeek -V3	Claude 3.5 Sonnet	Nova Pro 1.0
Simple Http Server	2	3	3	3	3
Reciept generator	2	2	1	3	1
Password	3	2	2	3	2
Search books...	2	2	2	2	1
Snake	1	1	2	3	2
Restaurant-Man...	3	2	2	2	2
Alarm clock	2	3	3	3	2
Health Log Book	1	2	2	2	2
Ball_bounce	2	3	3	3	2
Finance Tracker	1	2	1	1	1
Simple Calculator	3	3	3	3	3
Organizer	3	3	2	3	2
Notepad	1	0	0	2	0
Addressbook	2	3	2	3	3
Geometry	2	2	2	2	0
SudokuGUI	2	2	3	1	2
Scientific CalC	3	2	1	2	1
Pong	1	1	1	1	1^{1}
Billing system	1	0	1	2^{1}	1^{1}
Cafe-management	0	0	0	3	0
Total	37	38	37	46	31

10.16/10.60 with VAPU on average. With the highest OD values of 9-10 the values drop with both methods being 6.40 with ZSL and 8.20 with VAPU on average. As such, the values decrease with higher OD values, indicating an increase in difficulty.

OD was divided into tasks, LOC, and CC to illustrate the differences between each value for the amount of check marks, which are all visible in Table 4.15. In the table, the results of the task amounts are divided from one to three. With one task, ZSL the result was 11.50 and for VAPU 11.32 check marks on average. With two tasks, the result decreased to 8.00 with ZSL and 9.38 for VAPU on average. With the three tasks, both methods improved slightly with the ZSL value measured in 9.33 and VAPU with 10.17 on average. Therefore, the average value of both methods dropped from one task to two tasks but did increase slightly with three tasks.

Division of code length is illustrated with files under 100, between 100 and 300, and those over 300 LOC. With LOC under 100, ZSL had an average value of 11.86 and VAPU

Table 4.13. *Results of GitHub updates with VAPU by difficulty*

Project	GPT-4o mini	GPT-4o	DeepSeek -V3	Claude 3.5 Sonnet	Nova Pro 1.0
Simple Http Server	3	3	3	3	3
Reciept generator	2	2	1	3	1
Password	3	2	2	2	2
Search books...	3	2	3	3	3
Snake	1	2	1	3	2
Restaurant-Man...	3	2	2	2	2
Alarm clock	3	2	3	3	3
Health Log Book	1	2	1	2	1
Ball_bounce	2	3	3	2	2
Finance Tracker	1	3	1	3	1
Simple Calculator	2	3	2	3	3
Organizer	2	3	3	3	2
Notepad	1	2	1	2	1
Addressbook	1	2	3	2	2
Geometry	2	2	2	1	3
SudokuGUI	2	3	3	1	2
Scientific CalC	2	2	3	3	2
Pong	1	1	0	2	1^{1}
Billing system	1	2	1^{1}	0^{1}	2^{1}
Cafe-management	2	2	0	2	1
Total	38	45	38	45	38

12.43. With higher LOC the value of both ZSL and VAPU dropped to 9.33. With LOC over 300, ZSL value dropped to 7.14 while VAPU dropped to 8.71 on average. As such, both methods' values did drop with increased LOC but slightly lesser with VAPU.

CC is divided into letters A, B, and C-E. With the CC value of A the average ZSL value was 9.38 and 10.75 with VAPU. With CC increased to B, the average value of both methods increased with ZSL to 11.50 and VAPU to 10.66, With the highest levels of CC, ZSL decreased to 7.50 and VAPU to 9.00. Based on the findings, the CC of B was highest with both methods, with ZSL having a higher average value with B and VAPU higher with the rest.

When the three different difficulty metric are compared to each other, LOC is the only metric with the values fully regressing with the increased difficulty. This might indicate that LOC is the most influencing factor between the difficulties. With the ratio values, the increase in LOC from less than 100 to more than 350 LOC changed the 3.6% better

Table 4.14. *Total and average check marks by OD*

Method	3-4	5-6	7-8	9-10	Total
ZSL	44 (11.00)	61 (10.16)	52 (10.40)	32 (6.40)	189
VAPU	49 (12.25)	61 (10.16)	53 (10.60)	41 (8.20)	204

Table 4.15. *Total and average check marks with the each metric*

Method	**Task amount**			
	one task	**two tasks**	**three tasks**	**Total**
ZSL	69 (11.50)	64 (8.00)	56 (9.33)	189
VAPU	68 (11.33)	75 (9.38)	61 (10.17)	204
		LOC		
	<100 LOC	**100-300 LOC**	**>300 LOC**	**Total**
ZSL	83 (11.86)	56 (9.33)	50 (7.14)	189
VAPU	87 (12.43)	56 (9.33)	61 (8.71)	204
		CC		
	A	**B**	**C-E**	**Total**
ZSL	75 (9.38)	69 (11.50)	45 (7.50)	189
VAPU	86 (10.75)	64 (10.66)	54 (9.00)	204

performance of ZSL to 22.0% better performance for VAPU, which means a total change of 25.6%. With increasing CC from A to C-E, the ratio changed from 13.2% to 20.0% for VAPU. The task amount ratio changed from 3.0% benefit of ZSL in one task to 9.0% benefit for VAPU in three tasks. Based on the changes in performance ratio, LOC is also the most changing metric for explaining VAPU's better performance in high OD levels.

5. DISCUSSION

This chapter discusses the obtained results and analyses them to examine the research questions of the thesis. The chapter starts with LLM's impact on code updating and VAPU's performance. The second section discusses the temperature's impact on VAPU. The third section discusses VAPU's recognized challenges in autonomously updating applications. This is followed by the benefits of using VAPU. The next section discusses the limitations of the thesis. Last, the implications of the study and future work section discusses the future impacts of the topic.

5.1 LLMs Impact with Code Updating and to the Functionality of VAPU

In this thesis, six view files from a legacy web application were updated in different LLMs using both OSL/ZSL prompts and a multi-agent system named VAPU. The result demonstrates that an LLM does not need to be large to accomplish a simple version update for code files. In fact, the best results for version update tasks were accomplished with DeepSeek-V3 and Nova Pro 1.0. This indicates that a relatively small model can product fully updated code even better than a bigger alternative. A similar phenomenon was observed in Shojaee *et al*. [88], where comparison of reasoning models with LLMs showed that LLMs perform better in simpler problem-solving tasks despite reasoning models performing overall better in harder ones.

Especially DeepSeek-V3 had an impressive rate of zero fatal errors in view files from A to D in EDP and overall stable performance as indicated by a low standard deviation. The reason why DeepSeek-V3 gets overall impressive results might be because of its architecture. In the technical report of the model [61], it is stated that it uses multi-token prediction, and instead of predicting one token, it predicts two tokens. As stated in the background research, Gloeckle *et al*. [19] found improvements using multi-token prediction in coding tasks, which was used as motivation when making DeepSeek-V3 [61]. As other model architectures are not disclosed, it can not be certain if these models utilize this technique also. As such, a comparison of open-source models would be needed to confirm this.

LLM had a notable impact on the outcomes of both VAPU and compared OSL/ZSL

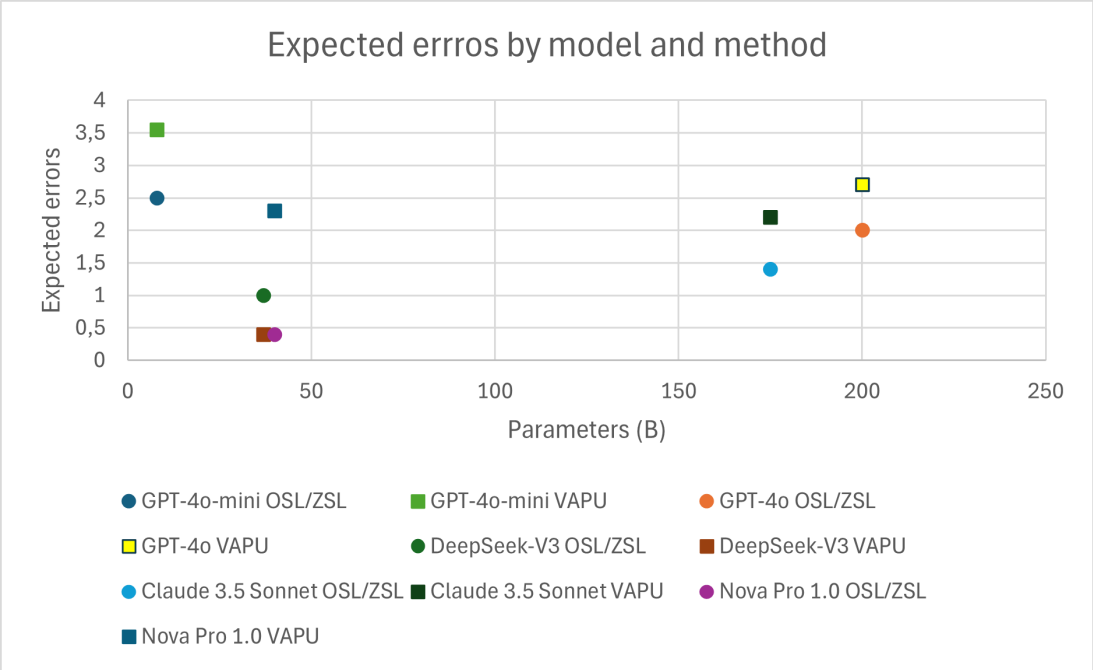


Figure 5.1. *Change of expected errors by parameter count*

prompts. When the best result between the compared temperatures is taken as seen in Figure 5.1 with the small LLM, the errors are high, which decrease to the medium sized models and then again increases with the larger models. The results for both methods were similar in most models, with VAPU having a slightly higher error count. However, Nova Pro 1.0 had a relatively high difference between the methods. Another outlier was DeepSeek-V3, which performed better in VAPU than OSL/ZSL prompts.

With more complex requirements in Views E-F the results ended up differently. The larger parameter models performed much better compared to the lesser parameter models. As seen in Figure 5.2, the best score of expected requirements stays low with the small and medium LLMs, but increases with the higher parameter models. One notable result was the low performance of VAPU with Claude 3.5 Sonnet compared to any other model. This is explained by the findings of Section 4.3. As LOC was much higher in Views E and F than the rest of the models using VAPU it can be concluded that VAPU overdid tasks in the model which eventually leaded to errors causing the requirements to fail. This highlights the fact that the same multi-agent system does not necessarily behave similarly when an LLM is switched to an alternative one. As such, testing of a multi-agent system must be conducted on each model separately to ensure the wanted behaviour.

The hallucination rate of each LLM seems to not have an impact at such a high level on the produced errors as the parameter size. As DeepSeek-V3 and Claude 3.5 Sonnet had the highest rate of hallucinations in the observed LLMs, their good performance does not indicate that a slightly higher level of hallucination is a deciding factor. With the VAPU's low performance with the requirements in Claude 3.5 Sonnet, there could be a link, but

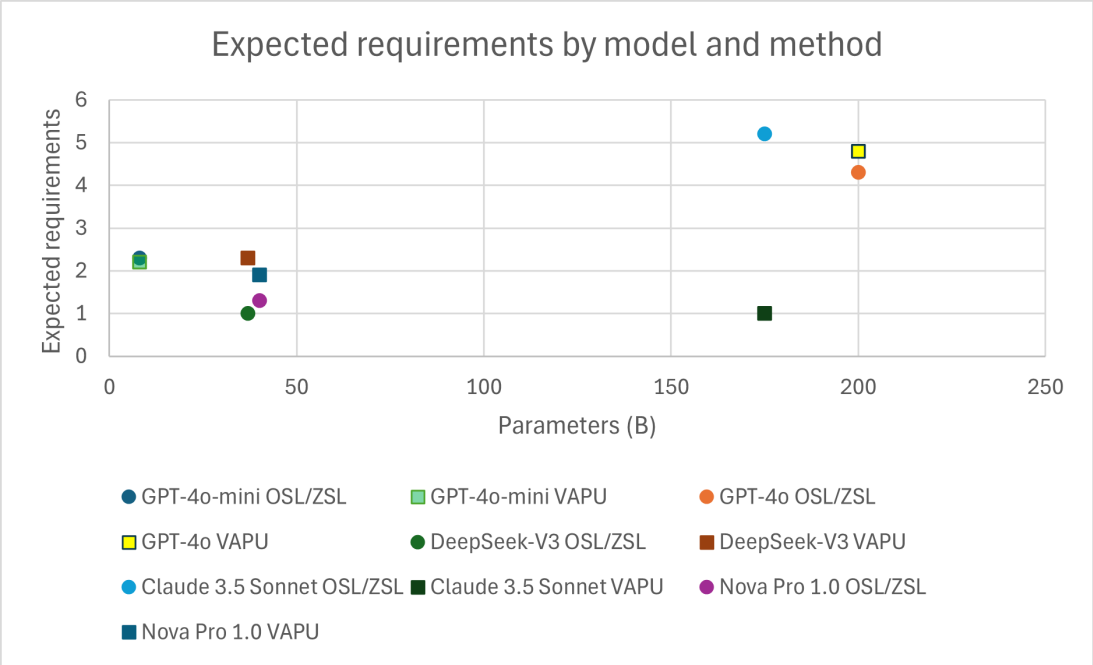


Figure 5.2. *Change of expected requirements by parameter count*

this would require a study with an alternative high-parameter model with a similar level of hallucination.

To answer LLMs impact on code updating and performance with the used multi-agent system, the following conclusion can be made. The increased parameter amount of an LLM adds the possibility of solving more complex code updating tasks with both multi-agent systems and prompts alone following the scaling laws of LLMs [4]. However, smaller parameter models might excel better in simpler version update tasks with correct architectural choices such as multi-token prediction [19, 61], and simpler models in general excel in simpler tasks [88]. LLM also impacts multi-agent systems behavior which might cause it to act differently compared to other models with updating tasks. Therefore, testing is required with each LLM separately to ensure multi-agent systems functionality.

5.2 Impact of Temperature to Code Updating and to the Functionality of VAPU

The impact of temperature was measured with the six studied view files in the average errors and the average requirements fulfilled in Section 4.2. As found in the results, lowering the temperature had a strong improvement for VAPU as opposed to OSL/ZSL prompts where this improvement did not happen similarly for each LLM. The rate of improvement was 33% reduction in errors and 80.0% increase in the requirements, illustrating a potential benefit, especially, as this was not observed with the OSL/ZSL prompts.

Compared to the results of the impact of the temperatures from the literature [32, 33]

which are discussed in Subsection 2.1.2 the average result of OSL/ZSL is in line to have only small significant impact for the results. Despite having a high variation of change of results in both directions in an individual LLM, the average ratio stays around the same. As such, changing temperature could improve results on a certain model and scenario but could as well lead to worse results as the result highlights. However, with VAPU there is a notable improvement especially in the requirement amount fulfilled. The results did improve in nine cases out of ten, which is a high contrast to the OSL/ZSL prompts, where they were distributed around equally.

Based on these results, temperature has a potential impact for multi-agent systems in code generation. Based on the different error types in Table 4.7 the improvement seems to impact in fatal, runtime, as content errors. As the reason why this improvement happens despite the standard deviation staying just as high as found in Sections 4.1 and 4.2 unlike in the OSL/ZSL prompts is an interesting subject. As the temperature does not go full deterministic for API based LLMs as seen for the existing standard deviations in Table 4.2 and Table 4.4, some variation is left for each request. As established in Equation 3.3 the outcome of a multi-agent system is formed from intermediate tokens, making the amount of tokens much higher than generating a simple OSL/ZSL prompt. As such, some variation is left for LLM-based multi-agent systems using API, due to a longer probability chain, as seen in Equation 3.3.

Despite the high standard deviation explained by the number of tokens, the improvement itself needs more investigation. One possible reason for the improvement could be that VAPU due to the amount of intermediate tokens has a broader outcomes of resulting code with higher temperatures, which includes more erroneous or faulty instances. With the temperature lowered, the likelihood answer concentrates more towards correct answers as the probability of incorrect answers decreases from the agents. Regardless, more information is needed to investigate whether the improvement is related to VAPU alone or if it occurs with multi-agent systems in general.

5.3 Challenges of Autonomously Updating Multi-agent System

During this thesis, one research question was the recognition of challenges encountered during the use of VAPU to update code files. The first challenge discussed is the amount of errors that VAPU produces. Even with a lower temperature, VAPU produced a high number of fatal and runtime errors. Compared to the literature, fatal and runtime errors are frequently produced by LLMs with runtime errors having a higher rate [48, 49]. This was also observed with a ratio of between 1/2-1/4 with OSL/ZSL prompts and 1/2 with VAPU at both temperatures. This finding leads to questions of the verifier agent's functionality, which was tested with a time analysis.

The time analysis in Section 4.3 showed that VAPU did not improve the results with longer

execution times but instead had a slight decrease in some cases. This indicates that the current verifier agent is not enough to find the remaining errors and the risks of the verifier hallucinating nonexistent errors are already encountered in the literature [48, 50], which could explain the decrease. For a multi-agent system to converge towards the desired outcome by producing the correct result, the verifier must be accurate and deterministic enough. Possible ways to improve the verifier are, for example, the following methods:

A reasoning verifier: Using a reasoning model might be able to solve some challenges regarding verification that the code fulfills the user requirements. Reasoning models are able to solve problems with much higher accuracy than LLM in problem-solving tasks [5]. However, reasoning models also seem to have a limit of problem-solving capacity before results collapse, as found in a recent study investigating the problem-solving capacity of reasoning models [88]. Therefore, a reasoning model could improve the results to a certain extent, but some errors would still be produced due to the same challenges as LLMs.

A dynamic test verifier: Multi-agent systems using test case verifying agents and tools have been used successfully to improve code quality, as found during the background study [2, 44]. When updating code, a multi-agent system could both ensure that the produced code complies with the I/O requirements from the previous version and any new requirements needed. Additionally, checking that the code can either be compiled or interpreted could remove fatal errors that happen when the code is tried to execute.

A static test verifier: The verification process could be carried out with different static testing methods, where the system would analyse the code without executing it. The combination of static testing tools and LLMs was investigated in Li *et al*. [89], which found that LLMs do not miss bugs found by a static tool, but some false errors can be found when LLM tries to solve harder tasks. Compared to a dynamic test verifier, a static test verifier could find errors that are hard to find, for example, in large code files, but have a possible risk of hallucination due to static testing producing false positives.

The amount of produced code was higher with VAPU than in OSL/ZSL prompts, especially with the Claude 3.5 sonnet. While some of the produced code was useful for the task, some of the results included unnessary content ranging from unnecessary library declarations to making some functions itself despite having a library alternative to do it. For example, the low number of replaced functions in View F illustrates this. Despite not counting the performance errors in this thesis, the outcome highlights that the system should be able to create better performing code to autonomously update files. As such, the system should include a checker to evaluate the performance. For example, Gao *et al*. [90] presented a Search-Based LLM (SBLLM), which combined evolutionary algorithms with LLMs to refactor given code to optimal versions. The multi-step system achieved increase of optimization compared to baseline between 1.15% and 28.06% and illustrates

the fact that LLM-generated code can be optimized with the right methods [90].

During the update of Python files, one more challenge was the limited output of the LLMs which at lowest was 5000 tokens. This creates a limitation of the updated file size which quickly becomes a challenge with multiple or larger files. To solve this challenge, multiple code files could be returned separately in the pipeline as modules, as in the CodePori system [41]. With the code files longer than the output, the system could save the produced code to the context length and request LLM to keep updating where it was left. The efficiency of such a method requires further investigation with VAPU.

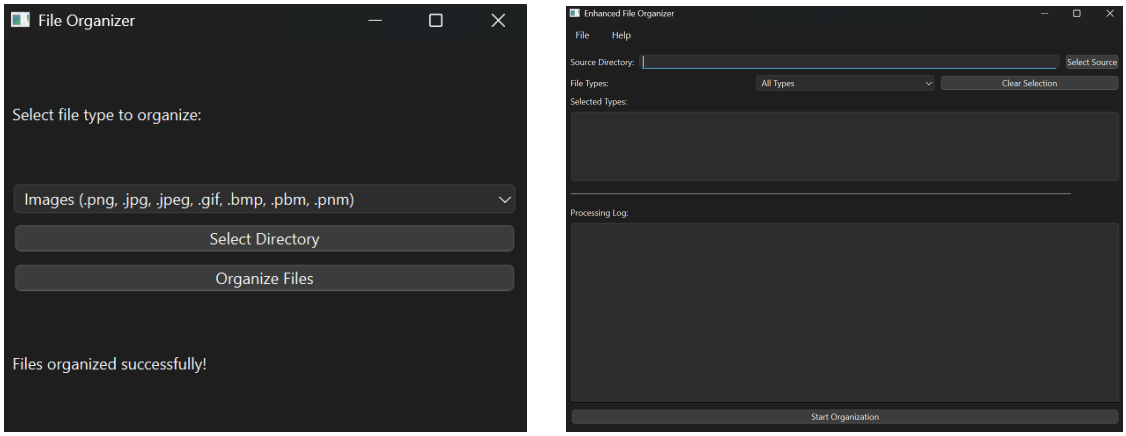
Overall, VAPU has challenges with the produced errors due to a underperforming verifier and is limited by code length that is sometimes produced unnecessary. Adding a better performing reasoning model that would have access to tools to test the code dynamically and statically could notably lower the amount of produced errors as found in the literature. With tools to improve performance and architectural changes to the system, VAPU could update multiple or longer code files with better optimized code.

5.4 Review of Functionality of VAPU

Based on the best results in VAPU obtained by lowering the temperature to zero in LLMs, VAPU shows promise with a more complex updating tasks in the electronic dictionary. In three models out of five, VAPU did produce better results on average, which can be seen in Figure 5.2. However, with simpler tasks, the error rate was higher with 0.71 errors on average, which illustrates slight challenges with producing error-free code. With VAPU updating the Python files in the GitHub repositories, VAPU excelled in two models with average check marks higher than the comparison by 22.5% and 18.4% respectfully.

When compared to the LLM-based systems that produces code the mean improvement rate varied between 7.3%-20.5% with the HumanEval metric as mentioned in Table 2.2. While the metric cannot be directly compared to the code updating in the Python files, the rate of improvement seems to be at the higher end in those LLMs which VAPU improved when compared to the percentages. As such, despite the challenges existing in the system discussed in the previous section, VAPU performs at the rate previously observed in the literature with the LLMs that were best compatible with this implementation of VAPU.

When the check marks were divided into the amount of tasks, the cyclomatic complexity, and code length, VAPU showed better performance with multiple tasks, longer code lengths, and with higher cyclomatic complexes. As stated in Section 3.2, one expected improvement of VAPU is the performance improvement in longer prompts due to dividing prompt into smaller tasks. An increase of 9.0% in performance with three tasks and 22.0% increase in files with LOC over 300 shows that the system succeeds with the intended purpose.



(a) organizer.py with ZSL (b) organizer.py with VAPU

Figure 5.3. *Comparison of organizer.py between ZSL and VAPU using Claude 3.5 Sonnet*

One possible benefit possibility of VAPU is the added quality output of files that could need an extension in quality. Figure 5.3 contains screenshots of the outcome of changing organizer.py from a terminal representation to PySide6 with Claude 3.5 Sonnet. With the same prompt, the resulted version with VAPU was more professional with the functionalities and the layout. While this was not the primary function of the system, variations of the system could be produced which would focus on quality enhancing features or security enhancements with additional agents.

Despite challenges observed with VAPU, the system performs better than the baseline in complex update tasks with certain LLMs. The improved LLMs had the same level of performance as the best performing LLMs in their parameter size group. The possibilities of extending the system give additional possibilities of increasing it's functionality from features to security.

5.5 Limitations

This thesis acknowledges limitations with the obtained results. As the verification process was done in six view files it is possible that the results could differ with more files added from the electronic dictionary or from a different source. As the study did not find a widely recognized metric for code updating and refactoring, a comparison to other studies is still needed to ensure validity similar to the HumanEval metric for code-generating multi-agent systems [25]. Additionally, more LLMs and different temperature values could give more or divergent outcomes. With the GitHub Python files, more files and tasks could give similarly different outcomes. The exact impacts of code length, CC, and the number of tasks for code generation in VAPU would need a more specific dataset for individual testing of variables.

The validity of the system's performance is another recognized limitation. Due to the

nature of LLM giving a probabilistic output as by Equation 2.3, the results change by input. As such, even a slight change in an agent's system prompt could have an improvement or a downgrade in the system's functionality. As such, testing VAPU with slight alternative ways should validate the architecture of the multi-agent system with a higher certainty. This is also correct for the prompts used that could have a similar impact on the results as seen in Equation 2.3. As such, more prompt engineering techniques could be included for the scope to validate the results from the prompts perspective for the comparison and for the multi-agent systems functionality in different requests.

5.6 Implications of the Study and Future Work

As LLMs improve with code updating and multi-agent systems become better refined, there are expected impacts to the industry. As mentioned in the introduction, between 40-90% of software companies resources are spent on maintaining applications [8, 9]. As seen in this thesis, simpler updating tasks have only handful of errors, therefore, using them in a non-critical updating tasks might be possible in very near future. With more complex tasks that involve the safety and functionality of critical systems, LLMs need to solve challenges such as defective code and that is not aligned with all client requirements. With the defective code, the possible solutions found in Section 5.3 could notably improve the rate of correct code. With the client's requirements, having a more structured system and correct LLMs or reasoning models could bring a higher level of certainty to the all requirements as already observed with VAPU in Section 5.4.

With improved multi-agent systems, updating entire applications might be possible to be implemented. While some level of hallucination is expected to persist in LLMs and reasoning models [38, 40, 88], having an autonomous system to do most of the work will free company resources from code maintenance. Many software engineers desire to use LLMs for code improvement and maintenance and use them for automation [91]. With the new systems, the job description of software engineers will then focus from code maintenance to instructing these updating systems and models, and addressing possible errors that are produced by them. More analysis is needed in the future to understand the implications of this change.

Opportunities also arise from transforming programs across coding languages by keeping their logical structure. As many programs still in use are written in legacy coding languages like COBOL, which was one of the legacy application case studies being translated into Java [53] in Section 2.4. There are studies trying to translate programs across coding languages [92, 93]. With GPT-4, the translation between modern languages like Python, Java, and C++ had at best a successful translation rate above 80% but averaging to short of 50% [92]. With an attempt to translate C and Go libraries into rust, the best translation rate was at 47% with GPT-4o, with a peak a bit above 80% in one library. As

with the new reasoning models and newly trained LLMs, these numbers are expected to rise higher.

However, current LLMs have strong difficulties in solving legacy programming language problems, with solving only a fraction of problems compared to modern languages. For example, with COBOL depending on the difficulty, only 0-23% of problems were solved in contrast to up to 89% success rate with Python on easier tasks [94]. This is believed to be due to the lack of training data in places such as GitHub repositories [94]. For LLM and multi-agent systems to be able to translate legacy programming languages, they need to be trained with more data related to legacy programming languages to successfully translate legacy applications to modern coding languages.

From the academic perspective this thesis provides beneficial findings. The improvement of using lower temperature focused on the multi-agent system provides opportunities to study the difference of impact in complex systems compared to simple calls to the LLM. The difference in success between the complexity of task and the parameter size of LLM provides support to further understanding the optimization of LLM to a certain task.

For future work a study creating an improved multi-agent system based on the thesis results is a possibility. For example, creating a more deterministic verifier for VAPU is a good consideration for future work to improve the system to make it produce results closer to the intended probability outcome where the results are inline with the requirements. Another possibility is to focus on the interaction between a client and the system like follow-up questions and clarifications. Additionally, looking for different alternative architectures for different agent roles and relations to obtain improved results is a possible path forward. With the results obtained from lowering the temperature of VAPU, a study confirming these findings with alternative multi-agent systems could be a possibility to validate the performance improving effect of lower temperatures in multi-agent systems.

6. CONCLUSION

In this thesis, a multi-agent system VAPU was used to study the capabilities of an LLM-based multi-agent system for updating legacy code and developing source code based on client's requirements. From a legacy electronic English dictionary [63], six view files were updated by comparing VAPU to OSL/ZSL prompts in different settings and multiple times to get the probability distribution to study the performance of the system. As an extension to the previous study Ala-Salmi *et al*. [13], the LLMs used were increased from one to five, with a comparison of temperatures between 0 and 1, and a verification of the system with 20 GitHub Python projects.

The findings of comparing LLMs showed differences between the models in different updating tasks. The medium parameter models had, at best, one recurring error less on average compared to the higher parameter models and two to the small parameter model. The higher parameter models, on the other hand, had over two and half times more accepted requirements on harder tasks than the lesser parameter models. For practitioners, this indicates that the best LLM for the system might not be the one with a high parameter count if the tasks are not complex enough.

Lowering of the temperature did have a notable change in performance with VAPU by reducing errors and improving the probability of correctly performing complex tasks. For an academic perspective, this indicates that a system using intermediate tokens to produce output might benefit from lower temperatures compared to methods producing output instantly.

The categorization of the produced errors revealed that VAPU produces mainly fatal and runtime errors, and while the temperature lowering did reduce them, a high number of them still persist in the generation. The time analysis of the system showed that the verifier-finalizer agent loopback did not improve the system when the system executed the updating process longer. Due to the underperforming verifier, a better verifier is needed with the help of a reasoning model, a test bench with test cases, or with different statistical test methods. As such, this thesis provides challenges and their possible solutions for practitioners to improve similar systems.

Updating 20 Python files showed performance improvements in two LLMs with the improvement rate rising up by 22.5%. The improvement lifted both models to competing

with the highest score in their own parameter class. This shows that a multi-agent system is a valid way to update code, with increased probability compared to the baseline in compatible LLMs. As such, multi-agent systems are a considerable way for companies to automatize parts of the code updating process and making the process more appealing to conduct.

Overall, this thesis has shown that a multi-agent system can autonomously update code with a rate of error similar to OSL/ZSL prompts but with a possibility of a higher amount of correct updates to the code. VAPU provides a foundation for future multi-agent systems that are designed to update code. For future work, a system capable of updating whole applications autonomously is an area of interest.

REFERENCES

[1] Adam Fourney et al. “Magentic-one: A generalist multi-agent system for solving complex tasks”. In: *arXiv preprint arXiv:2411.04468* (2024).

[2] Dong Huang et al. “Agentcoder: Multi-agent-based code generation with iterative testing and optimisation”. In: *arXiv preprint arXiv:2312.13010* (2023).

[3] Yoichi Ishibashi and Yoshimasa Nishimura. “Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization”. In: *arXiv preprint arXiv:2404.02183* (2024).

[4] Jared Kaplan et al. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).

[5] Daya Guo et al. “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning”. In: *arXiv preprint arXiv:2501.12948* (2025).

[6] Wendy Hall and Thanassis Tiropanis. “Web evolution and Web science”. In: *Computer Networks* 56.18 (2012), pp. 3859–3865.

[7] Nurullah Demir et al. “Our (in) secure web: understanding update behavior of websites and its impact on security”. In: *Passive and Active Measurement: 22nd International Conference, PAM 2021, Virtual Event, March 29–April 1, 2021, Proceedings 22*. Springer. 2021, pp. 76–92.

[8] Len Erlikh. “Leveraging legacy system dollars for e-business”. In: *IT professional* 2.3 (2000), pp. 17–23.

[9] Barbee Davis, ed. *97 things every project manager should know: collective wisdom from the experts*. 1. Aufl. Beijing Sebastopol, Calif.: O’Reilly, 2009. 223 pp. ISBN: 978-0-596-80416-9.

[10] Mubashir Ali et al. “Addressing Software Related Issues On Legacy Systems -A Review”. In: *International Journal of Scientific & Technology Research* 9 (Mar. 2020), pp. 3738–3742.

[11] Gábor Antal et al. “Transforming c++ 11 code to c++ 03 to support legacy compilation environments”. In: *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE. 2016, pp. 177–186.

[12] Sandra Smyth. “Penetration Testing and Legacy Systems”. In: *arXiv preprint arXiv:2402.10217* (2023).

[13] Valtteri Ala-Salmi et al. “Autonomous Legacy Web Application Upgrades Using a Multi-Agent System”. In: *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE*. IN-

STICC. SciTePress, 2025, pp. 185–196. ISBN: 978-989-758-742-9. DOI: 10.5220/0013397200003928.

[14] Valtteri Ala-Salmi. *Verifying Agent Pipeline Updater (VAPU)*. github.com. June 24, 2025. URL: https : / / github . com / alasalm1 / Multi - agent - pipeline (visited on 06/29/2025).

[15] Wayne Xin Zhao et al. “A survey of large language models”. In: *arXiv preprint arXiv:2303.18223* 1.2 (2023).

[16] Yao Fu et al. “Specializing smaller language models towards multi-step reasoning”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 10421–10430.

[17] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9.

[18] Julia Witte Zimmerman et al. “Tokens, the oft-overlooked appetizer: Large language models, the distributional hypothesis, and meaning”. In: *arXiv preprint arXiv:2412.10924* (2024).

[19] Fabian Gloeckle et al. “Better & faster large language models via multi-token prediction”. In: *arXiv preprint arXiv:2404.19737* (2024).

[20] Alec Radford and Karthik Narasimhan. “Improving Language Understanding by Generative Pre-Training”. In: 2018. URL: https://api.semanticscholar.org/CorpusID:49313245.

[21] Jacob Devlin et al. “Pre-training of deep bidirectional transformers for language understanding. arXiv”. In: *arXiv preprint arXiv:1810.04805* (2018).

[22] A Vaswani. “Attention is all you need”. In: *Advances in Neural Information Processing Systems* (2017).

[23] Maximilian Beck et al. “xlstm: Extended long short-term memory”. In: *arXiv preprint arXiv:2405.04517* (2024).

[24] Huan Ma et al. “Estimating LLM Uncertainty with Logits”. In: *arXiv preprint arXiv:2502.00290* (2025).

[25] Mark Chen et al. “Evaluating large language models trained on code”. In: *arXiv preprint arXiv:2107.03374* (2021).

[26] Banghao Chen et al. “Unleashing the potential of prompt engineering for large language models”. In: *Patterns* (2025).

[27] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Advances in neural information processing systems* 35 (2022), pp. 24824–24837.

[28] Tom B. Brown et al. “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165* (2020).

[29] Laria Reynolds and Kyle McDonell. “Prompt programming for large language models: Beyond the few-shot paradigm”. In: *Extended abstracts of the 2021 CHI conference on human factors in computing systems*. 2021, pp. 1–7.

[30] D.E. Koulouriotis and A. Xanthopoulos. “Reinforcement learning and evolutionary algorithms for non-stationary multi-armed bandit problems”. eng. In: *Applied mathematics and computation* 196.2 (2008), pp. 913–922. ISSN: 0096-3003.

[31] Honghua Chen and Nai Ding. “Probing the creativity of large language models: Can models produce divergent semantic association?” In: *arXiv preprint arXiv:2310.11158* (2023).

[32] Matthew Renze. “The effect of sampling temperature on problem solving in large language models”. In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. 2024, pp. 7346–7356.

[33] Max Peeperkorn et al. “Is temperature the creativity parameter of large language models?” In: *arXiv preprint arXiv:2405.00492* (2024).

[34] Jordan Hoffmann et al. “Training compute-optimal large language models”. In: *arXiv preprint arXiv:2203.15556* (2022).

[35] Niklas Muennighoff et al. “Crosslingual generalization through multitask finetuning”. In: *arXiv preprint arXiv:2211.01786* (2022).

[36] Nelson F Liu et al. “Lost in the middle: How language models use long contexts”. In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 157–173.

[37] Chenxin An et al. “Why Does the Effective Context Length of LLMs Fall Short?” In: *arXiv preprint arXiv:2410.18745* (2024).

[38] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. “Hallucination is inevitable: An innate limitation of large language models”. In: *arXiv preprint arXiv:2401.11817* (2024).

[39] Ziwei Ji et al. “Survey of hallucination in natural language generation”. In: *ACM computing surveys* 55.12 (2023), pp. 1–38.

[40] Sourav Banerjee, Ayushi Agarwal, and Saloni Singla. “Llms will always hallucinate, and we need to live with this”. In: *arXiv preprint arXiv:2409.05746* (2024).

[41] Zeeshan Rasheed et al. “CodePori: Large-Scale System for Autonomous Software Development Using Multi-Agent Technology”. In: *arXiv preprint arXiv:2402.01411* (2024).

[42] Jules White et al. “A prompt pattern catalog to enhance prompt engineering with chatgpt”. In: *arXiv preprint arXiv:2302.11382* (2023).

[43] Noah Shinn et al. “Reflexion: Language agents with verbal reinforcement learning”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[44] Li Zhong, Zilong Wang, and Jingbo Shang. “Ldb: A large language model debugger via verifying runtime execution step-by-step”. In: *arXiv preprint arXiv:2402.16906* (2024).

[45] Chao Lei et al. “Planning-Driven Programming: A Large Language Model Programming Workflow”. In: *arXiv preprint arXiv:2411.14503* (2024).

[46] OpenAI. *Model Release Notes*. openai.com. https://help.openai.com/en/articles/9624314-model-release-notes. June 10, 2025. (Visited on 06/18/2025).

[47] OpenAI. *Hello GPT-4o*. openai.com. May 13, 2024. URL: https://openai.com/index/hello-gpt-4o/ (visited on 06/18/2025).

[48] Yue Liu et al. "Refining chatgpt-generated code: Characterizing and mitigating code quality issues". In: *ACM Transactions on Software Engineering and Methodology* 33.5 (2024), pp. 1–26.

[49] Shihan Dou et al. "What's Wrong with Your Code Generated by Large Language Models? An Extensive Study". In: *arXiv preprint arXiv:2407.06153* (2024).

[50] Chun Jie Chong, Zhihao Yao, and Iulian Neamtiu. "Artificial-Intelligence Generated Code Considered Harmful: A Road Map for Secure and High-Quality Code Generation". In: *arXiv preprint arXiv:2409.19182* (2024).

[51] Ana Nunez et al. "Autosafecoder: A multi-agent framework for securing llm code generation through static analysis and fuzz testing". In: *arXiv preprint arXiv:2409.10737* (2024).

[52] Ian Sommerville. *Software engineering*. Tenth edition. Always learning. Boston Columbus Indianapolis New York San Francisco Hoboken Amsterdam Cape Town Dubai London: Pearson, 2016. 1 p. ISBN: 978-1-292-09614-8.

[53] Alessandro De Marco, Valentin Iancu, and Ira Asinofsky. "COBOL to Java and newspapers still get delivered". In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2018, pp. 583–586.

[54] Jonas Fritzsch et al. "Microservices migration in industry: intentions, strategies, and challenges". In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2019, pp. 481–490.

[55] Slavimir Lj Vesić and Duško Laković. "A Framework for Evaluating Legacy Systems–A Case Study". In: *Kultura polisa* 20.1 (2023), pp. 32–50.

[56] Yeming Wen et al. "Grounding data science code generation with input-output specifications". In: *arXiv preprint arXiv:2402.08073* (2024).

[57] J Diego Zamfirescu-Pereira et al. "Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts". In: *Proceedings of the 2023 CHI conference on human factors in computing systems*. 2023, pp. 1–21.

[58] OpenAI. *OpenAI developer platform*. platform.openai.com. May 22, 2025. URL: https://platform.openai.com/docs/overview (visited on 05/22/2025).

[59] OpenRouter. *The Unified Interface for LLMs*. openrouter.ai. May 22, 2025. URL: https://openrouter.ai/ (visited on 05/22/2025).

[60] Asma Ben Abacha et al. "Medec: A benchmark for medical error detection and correction in clinical notes". In: *arXiv preprint arXiv:2412.19260* (2024).

[61] Aixin Liu et al. "Deepseek-v3 technical report". In: *arXiv preprint arXiv:2412.19437* (2024).

[62] Vectara. *Hallucination Leaderboard*. github.com. Apr. 29, 2025. URL: https://github.com/vectara/hallucination-leaderboard (visited on 05/22/2025).

[63] Juhani Norri, Marko Junkkari, and Timo Poranen. “Digitization of data for a historical medical dictionary”. In: *Language Resources and Evaluation* 54 (2020), pp. 615–643.

[64] CakePHP. *Introduction to CakePHP*. CakePHP 1.1 Cookbook. https://book.cakephp.org/1.1/en/introduction-to-cakephp.html. Dec. 6, 2024. (Visited on 06/18/2025).

[65] Tim Koschuetzki. *Extra Hot: CakePHP 1.2 Stable is finally released!* debuggable. http://debuggable.com/posts/extra-hot-cakephp-1.2-stable-is-finally-released!:4954151c-f87c-434b-abbd-4e404834cda3. Dec. 26, 2008. (Visited on 06/18/2025).

[66] CakePHP. *CakePHP 4.5.0 Released*. Bakery. https://bakery.cakephp.org/2023/10/14/cakephp_450.html. Oct. 14, 2023. (Visited on 06/18/2025).

[67] CakePHP. *New CakePHP 5.2 Chiffon*. cakephp.orgm. June 11, 2025. URL: https://cakephp.org/ (visited on 06/11/2025).

[68] CakePHP. *Retrieving Data & Results Sets*. https://book.cakephp.org/3/en/orm/retrieving-data-and-resultsets.html. May 5, 2025. (Visited on 06/18/2025).

[69] Wendkûuni C Ouédraogo et al. “Large-scale, Independent and Comprehensive study of the power of LLMs for test case generation”. In: *arXiv preprint arXiv:2407.00225* (2024).

[70] Hanka Robovska. *Simple Calculator*. github.com. May 21, 2025. URL: https://github.com/hrosicka/SimpleCalculator?tab=readme-ov-file (visited on 05/25/2025).

[71] The algorithms. *The Algorithms - Python*. github.com. May 22, 2025. URL: https://github.com/TheAlgorithms/Python (visited on 05/25/2025).

[72] George Zhukov. *python-snake*. github.com. Oct. 21, 2022. URL: https://github.com/GeorgeZhukov/python-snake/tree/master (visited on 05/25/2025).

[73] Python-Mini-Projects. *Python-Mini-Projects*. github.com. June 2, 2024. URL: https://github.com/Python-World/python-mini-projects (visited on 05/27/2025).

[74] Python-Mini-Project. *Python-Mini-Project*. github.com. May 26, 2022. URL: https://github.com/ndleah/python-mini-project (visited on 05/27/2025).

[75] python-beginner-projects. *python-beginner-projects*. github.com. Feb. 11, 2025. URL: https://github.com/Mrinank-Bhowmick/python-beginner-projects (visited on 05/27/2025).

[76] Amazing-Python-Scripts. *Amazing-Python-Scripts*. github.com. June 5, 2024. URL: https://github.com/avinashkranjan/Amazing-Python-Scripts/tree/main (visited on 05/30/2025).

[77] Python-Projects. *Python-Projects*. github.com. Jan. 29, 2025. URL: https://github.com/Shahrayar123/Python-Projects/tree/master (visited on 05/30/2025).

[78] Python-GUI-Project. *Python-GUI-Project*. github.com. Mar. 9, 2025. URL: https://github.com/Aashishkumar123/Python-GUI-Project (visited on 05/30/2025).

[79] Michele Lacchia. *Command-line Usage*. radon.readthedocs.io. May 24, 2025. URL: https://radon.readthedocs.io/en/latest/commandline.html (visited on 01/23/2020).

[80] Python Software Foundation. *http.server — HTTP servers*. docs.python.org. June 11, 2025. URL: https://docs.python.org/3/library/http.server.html (visited on 06/11/2025).

[81] Berkeley. *Receipts and proofs of payment*. law.berkeley.edu. Feb. 11, 2025. URL: https://www.law.berkeley.edu/business-services/purchasing-and-reimbursement/getting-reimbursed/receipts-and-proofs-of-payment/ (visited on 05/28/2025).

[82] Riverbank Computing Limited. *What is PyQt?* riverbankcomputing.com. 2024. URL: https://www.riverbankcomputing.com/software/pyqt/ (visited on 06/19/2025).

[83] The Qt Company Ltd. *Qt for Python*. qt.io. 2025. URL: https://doc.qt.io/qtforpython-6/ (visited on 06/19/2025).

[84] The Matplotlib development team. *Matplotlib: Visualization with Python*. matplotlib.org. 2025. URL: https://matplotlib.org/ (visited on 06/19/2025).

[85] Plotly. *Plotly Open Source Graphing Library for Python*. plotly.com. 2025. URL: https://plotly.com/python/ (visited on 06/19/2025).

[86] The Kivy Authors. *Kivy: The Open Source Python App Development Framework.* kivy.org. 2024. URL: https://kivy.org/ (visited on 06/19/2025).

[87] Python community. *pyttsx3 2.98*. pypi.org. Sept. 27, 2024. URL: https://pypi.org/project/pyttsx3/ (visited on 06/11/2025).

[88] Parshin Shojaee et al. “The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity”. In: *arXiv preprint arXiv:2506.06941* (2025).

[89] Haonan Li et al. “Enhancing static analysis for practical bug detection: An llm-integrated approach”. In: *Proceedings of the ACM on Programming Languages* 8.OOPSLA1 (2024), pp. 474–499.

[90] Shuzheng Gao et al. “Search-based llms for code optimization”. In: *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society. 2024, pp. 254–266.

[91] Daniel Russo. “Navigating the complexity of generative ai adoption in software engineering”. In: *ACM Transactions on Software Engineering and Methodology* 33.5 (2024), pp. 1–50.

[92] Rangeet Pan et al. “Lost in translation: A study of bugs introduced by large language models while translating code”. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024, pp. 1–13.

[93] Hasan Ferit Eniser et al. “Towards translating real-world code with llms: A study of translating to rust”. In: *arXiv preprint arXiv:2405.11514* (2024).

[94] Álvaro Barbero Jiménez. “An evaluation of LLM code generation capabilities through graded exercises”. In: *arXiv e-prints* (2024), arXiv–2410.

APPENDIX A: VIEW FILES UPDATING RESULTS

This appendix contains the update results of each updated view file by temperature and an LLM as raw Excel data.

View files A, B, C and D evaluation process (Model: GPT 4o-mini, Temperature = 1)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	7	35
View_A_ZSL_run_2	0	0	0	0	0	6	35
View_A_ZSL_run_3	0	0	0	0	0	8	35
View_A_ZSL_run_4	0	0	0	0	0	11	35
View_A_ZSL_run_5	0	0	0	0	0	7	35
View_A_ZSL_run_6	0	0	0	0	0	7	35
View_A_ZSL_run_7	0	0	0	0	0	8	35
View_A_ZSL_run_8	0	0	0	0	0	7	35
View_A_ZSL_run_9	0	0	0	0	0	8	35
View_A_ZSL_run_10	0	0	0	0	0	8	35
Average	0	0	0	0	0	7,7	35
Standard Deviation (SD)	0	0	0	0	0	1,268857754	0
View_A_VAPU_run_1	0	0	0	0	0	63	37
View_A_VAPU_run_2	0	0	0	0	0	57	37
View_A_VAPU_run_3	1	0	0	0	1	56	37
View_A_VAPU_run_4	0	1	0	0	1	62	35
View_A_VAPU_run_5	0	0	0	0	0	50	35
View_A_VAPU_run_6	0	1	0	0	1	48	35
View_A_VAPU_run_7	0	0	0	0	0	46	35
View_A_VAPU_run_8	1	0	0	0	1	74	35
View_A_VAPU_run_9	0	1	0	0	1	59	35
View_A_VAPU_run_10	0	0	0	0	0	53	37
Average	0,2	0,3	0	0	0,5	56,8	35,8
Standard Deviation (SD)	0,4	0,458257569	0	0	0,5	7,884161338	0,979795897
View_B_OSL_run_1	0	0	0	0	0	3	23
View_B_OSL_run_2	0	0	0	0	0	4	22
View_B_OSL_run_3	1	0	0	0	1	3	22
View_B_OSL_run_4	0	1	0	0	1	6	22
View_B_OSL_run_5	0	0	0	0	0	4	23
View_B_OSL_run_6	2	0	0	0	2	15	22
View_B_OSL_run_7	0	0	0	0	0	4	22
View_B_OSL_run_8	0	0	0	0	0	5	22
View_B_OSL_run_9	0	0	0	0	0	3	22
View_B_OSL_run_10	0	0	0	0	0	3	22
Average	0,3	0,1	0	0	0,4	5	22,2
Standard Deviation (SD)	0,640312424	0,3	0	0	0,66332496	3,464101615	0,4
View_B_VAPU_run_1	0	1	0	0	1	165	28
View_B_VAPU_run_2	0	1	0	0	1	33	22
View_B_VAPU_run_3	0	1	0	0	1	35	23
View_B_VAPU_run_4	0	0	0	0	0	35	23
View_B_VAPU_run_5	1	0	0	0	1	44	23
View_B_VAPU_run_6	0	1	0	0	1	48	34
View_B_VAPU_run_7	0	0	0	0	0	51	22
View_B_VAPU_run_8	1	0	0	0	1	51	23
View_B_VAPU_run_9	0	0	0	0	0	38	22
View_B_VAPU_run_10	0	0	0	0	0	32	23
Average	0,2	0,4	0	0	0,6	53,2	24,3
Standard Deviation (SD)	0,4	0,489897949	0	0	0,48989795	37,90989317	3,634556369
View_C_OSL_run_1	1	0	0	0	1	7	57
View_C_OSL_run_2	0	0	0	0	0	6	56
View_C_OSL_run_3	2	0	0	0	2	9	58
View_C_OSL_run_4	0	1	0	0	1	7	57

View_C_OSL_run_5	0	0	0	0	0	8	52
View_C_OSL_run_6	0	1	0	0	1	7	58
View_C_OSL_run_7	0	0	0	0	0	9	54
View_C_OSL_run_8	0	1	0	0	1	6	58
View_C_OSL_run_9	0	1	0	0	1	8	58
View_C_OSL_run_10	0	0	0	0	0	7	60
Average	0,3	0,4	0	0	0,7	7,4	56,8
Standard Deviation (SD)	0,640312424	0,489897949	0	0	0,64031242	1,019803903	2,181742423
View_C_VAPU_run_1	1	0	0	0	1	47	58
View_C_VAPU_run_2	0	0	0	0	0	70	66
View_C_VAPU_run_3	0	0	0	0	0	59	64
View_C_VAPU_run_4	2	1	0	0	3	65	60
View_C_VAPU_run_5	1	0	0	0	1	66	58
View_C_VAPU_run_6	0	0	0	0	0	60	58
View_C_VAPU_run_7	0	2	0	0	2	69	58
View_C_VAPU_run_8	0	1	0	0	1	60	60
View_C_VAPU_run_9	0	1	0	0	1	54	60
View_C_VAPU_run_10	1	0	0	0	1	54	58
Average	0,5	0,5	0	0	1	60,4	60
Standard Deviation (SD)	0,670820393	0,670820393	0	0	0,89442719	6,945502142	2,683281573
View_D_ZSL_run_1	0	0	0	0	0	11	166
View_D_ZSL_run_2	1	0	0	0	1	10	168
View_D_ZSL_run_3	1	0	0	0	1	11	164
View_D_ZSL_run_4	0	0	0	0	0	10	148
View_D_ZSL_run_5	0	0	0	0	0	10	168
View_D_ZSL_run_6	0	0	0	0	0	12	165
View_D_ZSL_run_7	1	0	0	0	1	10	130
View_D_ZSL_run_8	0	0	0	0	0	11	163
View_D_ZSL_run_9	0	0	0	0	0	11	164
View_D_ZSL_run_10	0	0	0	0	0	11	164
Average	0,3	0	0	0	0,3	10,7	160
Standard Deviation (SD)	0,458257569	0	0	0	0,45825757	0,640312424	11,35781669
View_D_VAPU_run_1	0	1	0	0	1	68	173
View_D_VAPU_run_2	0	2	0	0	2	183	109
View_D_VAPU_run_3	0	1	0	0	1	73	163
View_D_VAPU_run_4	0	1	0	0	1	76	173
View_D_VAPU_run_5	0	1	0	0	1	59	164
View_D_VAPU_run_6	0	1	0	0	1	69	144
View_D_VAPU_run_7	0	3	0	0	3	57	145
View_D_VAPU_run_8	0	1	0	0	1	64	161
View_D_VAPU_run_9	0	0	0	0	0	99	171
View_D_VAPU_run_10	0	2	0	0	2	73	178
Average	0	1,3	0	0	1,3	82,1	158,1
Standard Deviation (SD)	0	0,781024968	0	0	0,78102497	35,39901129	19,63389926

View file E evaluation process (Model: GPT 4o-mini, Temperature = 1)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	0	1	0	1	36	4
View_E_ZSL_run_2	0	1	0	1	42	6
View_E_ZSL_run_3	0	0	0	0	46	5
View_E_ZSL_run_4	0	1	0	1	35	4
View_E_ZSL_run_5	0	0	0	0	62	5
View_E_ZSL_run_6	0	0	0	0	41	5
View_E_ZSL_run_7	0	1	0	1	46	5
View_E_ZSL_run_8	0	1	0	1	46	6
View_E_ZSL_run_9	0	0	0	0	40	5
View_E_ZSL_run_10	0	0	0	0	47	5
Average	0	0,5	0	0,5	44,1	5
View_E_VAPU_run_1	1	0	0	1	54	60
View_E_VAPU_run_2	0	0	0	0	47	52
View_E_VAPU_run_3	0	0	0	0	77	72
View_E_VAPU_run_4	0	0	0	0	49	73
View_E_VAPU_run_5	0	1	1	2	68	72
View_E_VAPU_run_6	1	0	0	1	95	81
View_E_VAPU_run_7	0	1	0	1	53	55
View_E_VAPU_run_8	1	1	0	2	51	45
View_E_VAPU_run_9	0	0	0	0	54	47
View_E_VAPU_run_10	0	0	0	0	116	93
average	0,3	0,3	0,1	0,7	66,4	65

View file F evaluation process (Model: GPT 4o-mini, Temperature = 1)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC
View_F_ZSL_run_1	1	1	1	3	4	35
View_F_ZSL_run_2	1	1	0	2	4	36
View_F_ZSL_run_3	0	0	0	0	4	32
View_F_ZSL_run_4	0	0	0	0	4	27

View_F_ZSL_run_5	1	1	1	3	1	66	
View_F_ZSL_run_6	1	1	1	3	4	35	
View_F_ZSL_run_7	1	1	1	3	1	72	
View_F_ZSL_run_8	1	0	0	1	2	43	
View_F_ZSL_run_9	1	1	1	3	4	35	
View_F_ZSL_run_10	0	0	0	0	4	37	
Avarage	0,7	0,6	0,5	1,8	3,2	41,8	
View_F_VAPU_run_1	0	0	0	0	4	44	
View_F_VAPU_run_2	0	0	0	0	4	40	
View_F_VAPU_run_3	1	0	0	1	2	50	
View_F_VAPU_run_4	1	0	1	2	0	77	
View_F_VAPU_run_5	1	0	1	2	3	41	
View_F_VAPU_run_6	0	0	0	0	2	51	
View_F_VAPU_run_7	0	0	0	0	4	35	
View_F_VAPU_run_8	0	0	0	0	1	82	
View_F_VAPU_run_9	0	0	0	0	3	46	
View_F_VAPU_run_10	0	0	0	0	4	57	
Avarage	0,3	0	0,2	0,5	2,7	52,3	

View files A, B, C and D evaluation process (Model: GPT 4o-mini, Temperature = 0)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	6	35
View_A_ZSL_run_2	0	0	0	0	0	7	35
View_A_ZSL_run_3	0	0	0	0	0	6	35
View_A_ZSL_run_4	0	0	0	0	0	6	35
View_A_ZSL_run_5	0	0	0	0	0	7	35
View_A_ZSL_run_6	0	0	0	0	0	7	35
View_A_ZSL_run_7	0	0	0	0	0	6	35
View_A_ZSL_run_8	0	0	0	0	0	6	35
View_A_ZSL_run_9	0	0	0	0	0	6	35
View_A_ZSL_run_10	0	0	0	0	0	6	35
Average	0	0	0	0	0	6,3	35
Standard Deviation (SD)	0	0	0	0	0	0,458257569	0
View_A_VAPU_run_1	0	0	0	0	0	58	35
View_A_VAPU_run_2	0	0	0	0	0	71	35
View_A_VAPU_run_3	2	0	0	0	2	74	35
View_A_VAPU_run_4	0	0	0	0	0	58	35
View_A_VAPU_run_5	0	0	0	0	0	53	35
View_A_VAPU_run_6	0	0	0	0	0	71	35
View_A_VAPU_run_7	0	0	0	0	0	51	35
View_A_VAPU_run_8	0	0	0	0	0	68	35
View_A_VAPU_run_9	0	0	0	0	0	58	35
View_A_VAPU_run_10	0	0	0	0	0	58	35
Average	0,2	0	0	0	0,2	62	35
Standard Deviation (SD)	0,6	0	0	0	0,6	7,797435476	0
View_B_OSL_run_1	0	1	0	0	1	5	23
View_B_OSL_run_2	0	0	0	0	0	7	23
View_B_OSL_run_3	0	1	0	0	1	6	23
View_B_OSL_run_4	0	1	0	0	1	15	22
View_B_OSL_run_5	0	1	0	0	1	6	23
View_B_OSL_run_6	0	1	0	0	1	7	23
View_B_OSL_run_7	0	1	0	0	1	6	23
View_B_OSL_run_8	0	1	0	0	1	7	23
View_B_OSL_run_9	0	1	0	0	1	6	23
View_B_OSL_run_10	0	1	0	0	1	6	23
Average	0	0,9	0	0	0,9	7,1	22,9
Standard Deviation (SD)	0	0,3	0	0	0,3	2,7	0,3
View_B_VAPU_run_1	0	0	0	0	0	57	23
View_B_VAPU_run_2	1	0	0	0	1	76	23
View_B_VAPU_run_3	0	1	0	0	1	64	25
View_B_VAPU_run_4	0	0	0	0	0	52	23
View_B_VAPU_run_5	0	0	0	0	0	321	27
View_B_VAPU_run_6	0	1	0	0	1	66	23
View_B_VAPU_run_7	0	0	0	0	0	53	22
View_B_VAPU_run_8	0	0	0	0	0	59	31
View_B_VAPU_run_9	0	0	0	0	0	53	23
View_B_VAPU_run_10	0	0	0	0	0	230	23
Average	0,1	0,2	0	0	0,3	103,1	24,3
Standard Deviation (SD)	0,3	0,4	0	0	0,45825757	88,83968708	2,60959767
View_C_OSL_run_1	3	0	0	0	3	15	56
View_C_OSL_run_2	3	0	0	0	3	10	56
View_C_OSL_run_3	3	0	0	0	3	10	56
View_C_OSL_run_4	3	0	0	0	3	9	56
View_C_OSL_run_5	3	0	0	0	3	13	56
View_C_OSL_run_6	3	0	0	0	3	11	56
View_C_OSL_run_7	3	0	0	0	3	13	56

View_C_OSL_run_8	3	0	0	0	3	13	56
View_C_OSL_run_9	3	0	0	0	3	12	56
View_C_OSL_run_10	3	0	0	0	3	13	56
Average	3	0	0	0	3	11,9	56
Standard Deviation (SD)	0	0	0	0	0	1,757839583	0
View_C_VAPU_run_1	0	1	0	0	1	85	59
View_C_VAPU_run_2	0	1	0	0	1	79	58
View_C_VAPU_run_3	0	2	0	0	2	88	58
View_C_VAPU_run_4	0	1	0	0	1	96	60
View_C_VAPU_run_5	0	1	0	0	1	116	64
View_C_VAPU_run_6	0	1	0	0	1	80	58
View_C_VAPU_run_7	0	1	0	0	1	84	58
View_C_VAPU_run_8	1	1	0	0	2	104	60
View_C_VAPU_run_9	0	1	0	0	1	88	64
View_C_VAPU_run_10	0	2	0	0	2	82	58
Average	0,1	1,2	0	0	1,3	90,2	59,7
Standard Deviation (SD)	0,3	0,4	0	0	0,45825757	11,23209687	2,282542442
View_D_ZSL_run_1	0	1	0	0	1	25	163
View_D_ZSL_run_2	0	1	0	0	1	24	164
View_D_ZSL_run_3	0	1	0	0	1	20	163
View_D_ZSL_run_4	0	1	0	0	1	26	163
View_D_ZSL_run_5	0	2	0	0	2	25	167
View_D_ZSL_run_6	0	2	0	0	2	20	167
View_D_ZSL_run_7	0	1	0	0	1	22	163
View_D_ZSL_run_8	0	2	0	0	2	25	169
View_D_ZSL_run_9	0	2	0	0	2	22	167
View_D_ZSL_run_10	0	2	0	0	2	25	167
Average	0	1,5	0	0	1,5	23,4	165,3
Standard Deviation (SD)	0	0,5	0	0	0,5	2,107130751	2,19317122
View_D_VAPU_run_1	0	1	0	0	1	132	163
View_D_VAPU_run_2	0	2	0	0	2	155	173
View_D_VAPU_run_3	0	2	0	0	2	147	167
View_D_VAPU_run_4	0	1	0	0	1	127	170
View_D_VAPU_run_5	0	1	0	0	1	189	163
View_D_VAPU_run_6	0	2	0	0	2	135	173
View_D_VAPU_run_7	0	2	0	0	2	127	167
View_D_VAPU_run_8	0	2	0	0	2	153	167
View_D_VAPU_run_9	0	1	0	0	1	122	163
View_D_VAPU_run_10	0	3	0	0	3	129	192
Average	0	1,7	0	0	1,7	141,6	169,8
Standard Deviation (SD)	0	0,640312424	0	0	0,64031242	19,21041384	8,219489035

View file E evaluation process (Model: GPT 4o-mini, Temperature = 0)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	0	1	0	1	36	4
View_E_ZSL_run_2	0	1	0	1	42	6
View_E_ZSL_run_3	0	0	0	0	46	5
View_E_ZSL_run_4	0	1	0	1	35	4
View_E_ZSL_run_5	0	0	0	0	62	5
View_E_ZSL_run_6	0	0	0	0	41	5
View_E_ZSL_run_7	0	1	0	1	46	5
View_E_ZSL_run_8	0	1	0	1	46	6
View_E_ZSL_run_9	0	0	0	0	40	5
View_E_ZSL_run_10	0	0	0	0	47	5
Average	0	0,5	0	0,5	44,1	5
View_E_VAPU_run_1	0	1	0	1	75	56
View_E_VAPU_run_2	0	0	0	0	54	47
View_E_VAPU_run_3	0	1	0	1	42	62
View_E_VAPU_run_4	0	1	0	1	44	44
View_E_VAPU_run_5	0	1	0	1	58	56
View_E_VAPU_run_6	0	0	0	0	51	42
View_E_VAPU_run_7	0	1	0	1	49	48
View_E_VAPU_run_8	0	1	0	1	50	62
View_E_VAPU_run_9	0	1	0	1	57	48
View_E_VAPU_run_10	0	1	0	1	46	70
average	0	0,8	0	0,8	52,6	53,5

View file F evaluation process (Model: GPT 4o-mini, Temperature = 0)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC
View_F_ZSL_run_1	0	0	0	0	4	32
View_F_ZSL_run_2	0	0	0	0	4	32
View_F_ZSL_run_3	0	0	0	0	4	32
View_F_ZSL_run_4	0	0	0	0	4	32
View_F_ZSL_run_5	0	0	0	0	4	32
View_F_ZSL_run_6	0	0	0	0	4	32
View_F_ZSL_run_7	0	0	0	0	4	32

View_F_ZSL_run_8	0	0	0	0	4	32	
View_F_ZSL_run_9	0	0	0	0	4	32	
View_F_ZSL_run_10	0	0	0	0	4	32	
Avarage	0	0	0	0	4	32	
View_F_VAPU_run_1	1	1	1	3	0	78	
View_F_VAPU_run_2	0	0	0	0	3	40	
View_F_VAPU_run_3	1	1	1	3	0	80	
View_F_VAPU_run_4	1	1	0	2	0	72	
View_F_VAPU_run_5	0	0	0	0	1	70	
View_F_VAPU_run_6	0	0	0	0	0	62	
View_F_VAPU_run_7	1	1	1	3	0	73	
View_F_VAPU_run_8	1	1	1	3	0	87	
View_F_VAPU_run_9	0	0	0	0	0	73	
View_F_VAPU_run_10	0	0	0	0	1	48	
Avarage	0,5	0,5	0,4	1,4	0,5	68,3	

View files A, B, C and D evaluation process (Model: GPT 4o, Temperature = 1)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	10	35
View_A_ZSL_run_2	1	0	1	0	2	6	32
View_A_ZSL_run_3	0	0	0	0	0	6	35
View_A_ZSL_run_4	0	0	0	0	0	5	36
View_A_ZSL_run_5	0	0	0	0	0	6	35
View_A_ZSL_run_6	0	0	1	0	1	6	32
View_A_ZSL_run_7	0	0	0	0	0	6	35
View_A_ZSL_run_8	0	0	0	0	0	7	35
View_A_ZSL_run_9	0	0	1	0	1	7	32
View_A_ZSL_run_10	0	0	0	0	0	7	36
Average	0,1	0	0,3	0	0,4	6,6	34,3
Standard Deviation (SD)	0,3	0	0,458257569	0	0,66332496	1,280624847	1,55241747
View_A_VAPU_run_1	1	0	0	0	1	38	35
View_A_VAPU_run_2	0	0	0	0	0	34	35
View_A_VAPU_run_3	0	0	0	0	0	24	37
View_A_VAPU_run_4	1	0	0	0	1	37	36
View_A_VAPU_run_5	2	0	0	0	2	31	31
View_A_VAPU_run_6	0	0	0	0	0	28	37
View_A_VAPU_run_7	0	0	0	0	0	33	37
View_A_VAPU_run_8	0	0	0	0	0	20	32
View_A_VAPU_run_9	0	0	0	0	0	30	37
View_A_VAPU_run_10	1	0	0	0	1	34	34
Average	0,5	0	0	0	0,5	30,9	35,1
Standard Deviation (SD)	0,670820393	0	0	0	0,67082039	5,356304696	2,071231518
View_B_OSL_run_1	0	0	0	0	0	11	24
View_B_OSL_run_2	0	0	0	0	0	10	25
View_B_OSL_run_3	0	0	0	0	0	9	23
View_B_OSL_run_4	0	0	0	0	0	10	23
View_B_OSL_run_5	0	0	0	0	0	9	25
View_B_OSL_run_6	0	0	0	0	0	9	23
View_B_OSL_run_7	0	0	0	0	0	5	22
View_B_OSL_run_8	0	0	0	0	0	7	26
View_B_OSL_run_9	0	0	0	0	0	5	24
View_B_OSL_run_10	0	0	0	0	0	6	23
Average	0	0	0	0	0	8,1	23,8
Standard Deviation (SD)	0	0	0	0	0	2,071231518	1,166190379
View_B_VAPU_run_1	0	0	0	0	0	38	30
View_B_VAPU_run_2	0	0	0	0	0	43	27
View_B_VAPU_run_3	0	1	0	0	1	32	39
View_B_VAPU_run_4	0	0	0	0	0	31	40
View_B_VAPU_run_5	0	0	0	0	0	32	28
View_B_VAPU_run_6	0	0	0	0	0	36	30
View_B_VAPU_run_7	1	0	0	0	1	40	43
View_B_VAPU_run_8	0	0	0	0	0	35	31
View_B_VAPU_run_9	0	0	0	0	0	34	25
View_B_VAPU_run_10	0	0	0	0	0	31	27
Average	0,1	0,1	0	0	0,2	35,2	32
Standard Deviation (SD)	0,3	0,3	0	0	0,4	3,867815921	5,983310121
View_C_OSL_run_1	1	0	0	0	1	8	58
View_C_OSL_run_2	0	0	0	0	0	7	65
View_C_OSL_run_3	1	0	0	0	1	6	60
View_C_OSL_run_4	1	0	0	0	1	6	56
View_C_OSL_run_5	0	0	0	0	0	5	64
View_C_OSL_run_6	0	0	0	0	0	6	57
View_C_OSL_run_7	0	0	0	0	0	11	60
View_C_OSL_run_8	0	0	0	0	0	7	72
View_C_OSL_run_9	0	0	0	0	0	11	58
View_C_OSL_run_10	0	0	0	0	0	7	62

Average	0,3	0	0	0	0,3	7,4	61,2
Standard Deviation (SD)	0,458257569	0	0	0	0,45825757	1,959591794	4,5563143
View_C_VAPU_run_1	0	1	0	0	1	52	58
View_C_VAPU_run_2	0	0	0	0	0	40	60
View_C_VAPU_run_3	0	0	0	0	0	31	58
View_C_VAPU_run_4	0	0	0	0	0	49	74
View_C_VAPU_run_5	0	0	0	0	0	37	52
View_C_VAPU_run_6	0	0	0	0	0	41	58
View_C_VAPU_run_7	0	1	0	0	1	42	63
View_C_VAPU_run_8	0	1	0	0	1	54	68
View_C_VAPU_run_9	0	0	0	0	0	40	71
View_C_VAPU_run_10	0	1	0	0	1	42	64
Average	0	0,4	0	0	0,4	42,8	62,6
Standard Deviation (SD)	0	0,489897949	0	0	0,48989795	6,64529909	6,437390776
View_D_ZSL_run_1	0	1	0	0	1	13	156
View_D_ZSL_run_2	0	2	0	0	2	14	162
View_D_ZSL_run_3	0	2	0	0	2	12	141
View_D_ZSL_run_4	0	2	0	0	2	14	165
View_D_ZSL_run_5	0	2	0	0	2	18	166
View_D_ZSL_run_6	0	0	0	0	0	13	141
View_D_ZSL_run_7	0	2	0	0	2	14	169
View_D_ZSL_run_8	0	1	0	0	1	14	162
View_D_ZSL_run_9	0	1	0	0	1	12	154
View_D_ZSL_run_10	0	1	0	0	1	14	136
Average	0	1,4	0	0	1,4	13,8	155,2
Standard Deviation (SD)	0	0,663324958	0	0	0,66332496	1,6	11,26765282
View_D_VAPU_run_1	1	1	0	0	2	68	159
View_D_VAPU_run_2	0	1	0	0	1	66	124
View_D_VAPU_run_3	0	2	0	0	2	84	173
View_D_VAPU_run_4	0	1	0	0	1	60	172
View_D_VAPU_run_5	0	2	0	0	2	73	162
View_D_VAPU_run_6	0	0	0	0	0	72	163
View_D_VAPU_run_7	1	1	0	0	2	57	157
View_D_VAPU_run_8	0	3	0	0	3	62	169
View_D_VAPU_run_9	1	2	0	0	3	48	177
View_D_VAPU_run_10	0	0	0	0	0	79	177
Average	0,3	1,3	0	0	1,6	66,9	163,3
Standard Deviation (SD)	0,458257569	0,9	0	0	1,0198039	10,15332458	14,77193285

View file E evaluation process (Model: GPT 4o, Temperature = 1)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	0	0	0	0	36	13
View_E_ZSL_run_2	0	0	0	0	37	15
View_E_ZSL_run_3	1	1	1	3	41	8
View_E_ZSL_run_4	1	0	0	1	41	17
View_E_ZSL_run_5	0	0	0	0	59	14
View_E_ZSL_run_6	0	1	0	1	36	8
View_E_ZSL_run_7	1	0	0	1	41	10
View_E_ZSL_run_8	1	0	0	1	35	7
View_E_ZSL_run_9	1	1	1	3	48	11
View_E_ZSL_run_10	1	0	0	1	41	9
Average	0,6	0,3	0,2	1,1	41,5	11,2
View_E_VAPU_run_1	0	0	0	0	37	48
View_E_VAPU_run_2	1	1	0	2	44	39
View_E_VAPU_run_3	0	1	0	1	44	70
View_E_VAPU_run_4	1	1	0	2	57	47
View_E_VAPU_run_5	1	0	0	1	43	42
View_E_VAPU_run_6	1	1	0	2	44	37
View_E_VAPU_run_7	1	0	0	1	37	59
View_E_VAPU_run_8	0	0	0	0	37	39
View_E_VAPU_run_9	1	0	0	1	45	51
View_E_VAPU_run_10	0	0	0	0	36	42
average	0,6	0,4	0	1	42,4	47,4

View file F evaluation process (Model: GPT 4o, Temperature = 1)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC	Time
View_F_ZSL_run_1	1	1	1	3	4	35	3
View_F_ZSL_run_2	1	1	1	3	4	34	3
View_F_ZSL_run_3	1	1	1	3	1	74	8
View_F_ZSL_run_4	1	1	0	2	4	37	3
View_F_ZSL_run_5	1	1	0	2	4	32	3
View_F_ZSL_run_6	0	0	0	0	2	75	6
View_F_ZSL_run_7	1	1	1	3	2	77	6
View_F_ZSL_run_8	1	1	1	3	2	57	5
View_F_ZSL_run_9	1	1	0	2	4	33	3
View_F_ZSL_run_10	0	0	0	0	4	33	3

Avarage	0,8	0,8	0,5	2,1	3,1	48,7	4,3
View_F_VAPU_run_1	0	0	0	0	0	67	39
View_F_VAPU_run_2	0	0	0	0	0	99	39
View_F_VAPU_run_3	0	0	0	0	3	61	28
View_F_VAPU_run_4	0	0	0	0	2	65	29
View_F_VAPU_run_5	0	0	0	0	4	37	28
View_F_VAPU_run_6	1	1	1	3	2	77	31
View_F_VAPU_run_7	0	0	0	0	1	67	32
View_F_VAPU_run_8	0	0	0	0	0	104	40
View_F_VAPU_run_9	0	0	0	0	1	78	36
View_F_VAPU_run_10	1	1	1	3	1	72	33
Avarage	0,2	0,2	0,2	0,6	1,4	72,7	33,5

View files A, B, C and D evaluation process (Model: GPT 4o, Temperature = 0)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	4	35
View_A_ZSL_run_2	0	0	0	0	0	4	35
View_A_ZSL_run_3	0	0	0	0	0	7	35
View_A_ZSL_run_4	0	0	0	0	0	4	35
View_A_ZSL_run_5	0	0	0	0	0	4	35
View_A_ZSL_run_6	0	0	0	0	0	5	35
View_A_ZSL_run_7	0	0	0	0	0	4	35
View_A_ZSL_run_8	0	0	0	0	0	4	35
View_A_ZSL_run_9	0	0	0	0	0	5	35
View_A_ZSL_run_10	0	0	0	0	0	4	35
Average	0	0	0	0	0	4,5	35
Standard Deviation (SD)	0	0	0	0	0	0,921954446	0
View_A_VAPU_run_1	0	0	0	0	0	23	35
View_A_VAPU_run_2	0	0	0	0	0	21	35
View_A_VAPU_run_3	0	0	0	0	0	22	35
View_A_VAPU_run_4	0	0	0	0	0	23	35
View_A_VAPU_run_5	2	0	0	0	2	23	35
View_A_VAPU_run_6	0	0	0	0	0	22	35
View_A_VAPU_run_7	2	0	0	0	2	24	33
View_A_VAPU_run_8	0	0	0	0	0	27	35
View_A_VAPU_run_9	2	0	0	0	2	24	35
View_A_VAPU_run_10	0	0	0	0	0	26	35
Average	0,6	0	0	0	0,6	23,5	34,8
Standard Deviation (SD)	0,916515139	0	0	0	0,91651514	1,74642492	0,6
View_B_OSL_run_1	0	0	0	0	0	71	23
View_B_OSL_run_2	0	0	0	0	0	8	23
View_B_OSL_run_3	0	0	0	0	0	9	23
View_B_OSL_run_4	0	0	0	0	0	10	23
View_B_OSL_run_5	0	0	0	0	0	8	23
View_B_OSL_run_6	0	0	0	0	0	7	23
View_B_OSL_run_7	0	0	0	0	0	6	23
View_B_OSL_run_8	0	0	0	0	0	6	23
View_B_OSL_run_9	0	0	0	0	0	6	23
View_B_OSL_run_10	0	0	0	0	0	6	23
Average	0	0	0	0	0	13,7	23
Standard Deviation (SD)	0	0	0	0	0	19,14706244	0
View_B_VAPU_run_1	0	0	0	0	0	57	28
View_B_VAPU_run_2	0	0	0	0	0	33	27
View_B_VAPU_run_3	0	0	0	0	0	37	27
View_B_VAPU_run_4	0	0	0	0	0	34	28
View_B_VAPU_run_5	0	0	0	0	0	40	**27**
View_B_VAPU_run_6	0	0	0	0	0	40	27
View_B_VAPU_run_7	0	0	0	0	0	32	30
View_B_VAPU_run_8	0	0	0	0	0	39	27
View_B_VAPU_run_9	0	0	0	0	0	40	28
View_B_VAPU_run_10	0	0	0	0	0	64	27
Average	0	0	0	0	0	41,6	27,6
Standard Deviation (SD)	0	0	0	0	0	9,991996797	0,916515139
View_C_OSL_run_1	0	0	0	0	0	5	58
View_C_OSL_run_2	0	0	0	0	0	5	56
View_C_OSL_run_3	0	0	0	0	0	4	56
View_C_OSL_run_4	0	0	0	0	0	7	58
View_C_OSL_run_5	0	0	0	0	0	6	58
View_C_OSL_run_6	0	0	0	0	0	7	58
View_C_OSL_run_7	0	0	0	0	0	8	58
View_C_OSL_run_8	0	0	0	0	0	6	58
View_C_OSL_run_9	0	0	0	0	0	9	56
View_C_OSL_run_10	0	0	0	0	0	10	58
Average	0	0	0	0	0	6,7	57,4
Standard Deviation (SD)	0	0	0	0	0	1,791647287	0,916515139

View_C_VAPU_run_1	0	0	0	0	0	36	67
View_C_VAPU_run_2	0	0	0	0	0	34	67
View_C_VAPU_run_3	0	1	0	0	1	37	64
View_C_VAPU_run_4	0	0	0	0	0	57	67
View_C_VAPU_run_5	0	1	0	0	1	38	64
View_C_VAPU_run_6	0	1	0	0	1	58	69
View_C_VAPU_run_7	0	1	0	0	1	42	64
View_C_VAPU_run_8	0	1	0	0	1	40	64
View_C_VAPU_run_9	0	1	0	0	1	62	64
View_C_VAPU_run_10	0	0	0	0	0	33	67
Average	0	0,6	0	0	0,6	43,7	65,7
Standard Deviation (SD)	0	0,489897949	0	0	0,48989795	10,38315944	1,791647287
View_D_ZSL_run_1	0	2	0	0	2	17	167
View_D_ZSL_run_2	0	2	0	0	2	15	167
View_D_ZSL_run_3	0	2	0	0	2	16	167
View_D_ZSL_run_4	0	2	0	0	2	20	167
View_D_ZSL_run_5	0	2	0	0	2	16	167
View_D_ZSL_run_6	0	2	0	0	2	22	167
View_D_ZSL_run_7	0	2	0	0	2	14	167
View_D_ZSL_run_8	0	2	0	0	2	13	167
View_D_ZSL_run_9	0	2	0	0	2	14	167
View_D_ZSL_run_10	0	2	0	0	2	14	167
Average	0	2	0	0	2	16,1	167
Standard Deviation (SD)	0	0	0	0	0	2,736786437	0
View_D_VAPU_run_1	0	1	0	0	1	47	168
View_D_VAPU_run_2	0	2	0	0	2	35	167
View_D_VAPU_run_3	0	2	0	0	2	32	170
View_D_VAPU_run_4	0	3	0	0	3	41	174
View_D_VAPU_run_5	0	2	0	0	2	34	173
View_D_VAPU_run_6	0	2	0	0	2	46	179
View_D_VAPU_run_7	0	2	0	0	2	44	173
View_D_VAPU_run_8	0	2	0	0	2	50	167
View_D_VAPU_run_9	0	2	0	0	2	41	172
View_D_VAPU_run_10	0	1	0	0	1	41	173
Average	0	1,9	0	0	1,9	41,1	171,6
Standard Deviation (SD)	0	0,538516481	0	0	0,53851648	5,629387178	3,527038418

View file E evaluation process (Model: GPT 4o, Temperature = 0)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	1	1	0	2	37	5
View_E_ZSL_run_2	1	0	0	1	36	4
View_E_ZSL_run_3	1	1	0	2	37	5
View_E_ZSL_run_4	1	0	1	2	37	7
View_E_ZSL_run_5	1	0	1	2	37	5
View_E_ZSL_run_6	1	1	1	3	37	4
View_E_ZSL_run_7	1	0	0	1	36	4
View_E_ZSL_run_8	1	0	0	1	37	5
View_E_ZSL_run_9	1	0	0	1	36	4
View_E_ZSL_run_10	1	0	0	1	37	8
Average	1	0,3	0,3	1,6	36,7	5,1
View_E_VAPU_run_1	1	1	0	2	42	41
View_E_VAPU_run_2	1	1	0	2	37	34
View_E_VAPU_run_3	1	1	0	2	37	33
View_E_VAPU_run_4	1	1	1	3	67	31
View_E_VAPU_run_5	1	1	0	2	50	26
View_E_VAPU_run_6	1	1	0	2	63	45
View_E_VAPU_run_7	1	1	0	2	45	34
View_E_VAPU_run_8	0	0	0	0	37	35
View_E_VAPU_run_9	1	1	0	2	37	25
View_E_VAPU_run_10	1	1	0	2	45	26
average	0,9	0,9	0,1	1,9	46	33

View file F evaluation process (Model: GPT 4o, Temperature = 0)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC	Time
View_F_ZSL_run_1	1	1	1	3	1	76	
View_F_ZSL_run_2	1	1	1	3	1	76	
View_F_ZSL_run_3	1	1	1	3	1	76	
View_F_ZSL_run_4	1	1	1	3	1	76	
View_F_ZSL_run_5	0	0	0	0	4	35	
View_F_ZSL_run_6	1	1	1	3	1	76	
View_F_ZSL_run_7	1	1	1	3	1	76	
View_F_ZSL_run_8	1	1	1	3	1	76	
View_F_ZSL_run_9	1	1	1	3	1	76	
View_F_ZSL_run_10	1	1	1	3	1	76	
Avarage	0,9	0,9	0,9	2,7	1,3	71,9	
View_F_VAPU_run_1	1	1	1	3	2	70	

View_F_VAPU_run_2	1	1	1	3	1	74	
View_F_VAPU_run_3	1	1	1	3	1	76	
View_F_VAPU_run_4	1	1	0	2	1	80	
View_F_VAPU_run_5	1	1	1	3	2	68	
View_F_VAPU_run_6	1	1	1	3	2	70	
View_F_VAPU_run_7	1	1	1	3	1	76	
View_F_VAPU_run_8	1	1	1	3	2	68	
View_F_VAPU_run_9	1	1	1	3	3	80	
View_F_VAPU_run_10	1	1	1	3	1	76	
Avarage	1	1	0,9	2,9	1,6	73,8	

View files A, B, C and D evaluation process (Model: DeepSeek-V3, Temperature = 1)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	40	35
View_A_ZSL_run_2	0	0	0	0	0	27	35
View_A_ZSL_run_3	0	0	0	0	0	26	35
View_A_ZSL_run_4	0	0	0	0	0	50	35
View_A_ZSL_run_5	0	0	0	0	0	29	35
View_A_ZSL_run_6	0	0	0	0	0	30	35
View_A_ZSL_run_7	0	0	0	0	0	41	35
View_A_ZSL_run_8	0	0	0	0	0	47	35
View_A_ZSL_run_9	0	0	0	0	0	25	35
View_A_ZSL_run_10	0	0	0	0	0	25	35
Average	0	0	0	0	0	34	35
Standard Deviation (SD)	0	0	0	0	0	9,088454214	0
View_A_VAPU_run_1	0	0	0	0	0	68	35
View_A_VAPU_run_2	0	0	0	0	0	60	35
View_A_VAPU_run_3	0	0	0	0	0	64	35
View_A_VAPU_run_4	0	0	0	0	0	76	35
View_A_VAPU_run_5	0	0	0	0	0	77	35
View_A_VAPU_run_6	0	0	0	0	0	74	35
View_A_VAPU_run_7	0	0	0	0	0	69	35
View_A_VAPU_run_8	0	0	0	0	0	66	35
View_A_VAPU_run_9	0	0	0	0	0	51	35
View_A_VAPU_run_10	0	0	0	0	0	70	35
Average	0	0	0	0	0	67,5	35
Standard Deviation (SD)	0	0	0	0	0	7,459892761	0
View_B_OSL_run_1	0	0	0	0	0	6	23
View_B_OSL_run_2	0	0	0	0	0	7	23
View_B_OSL_run_3	0	0	0	0	0	6	23
View_B_OSL_run_4	0	0	0	0	0	6	23
View_B_OSL_run_5	0	0	0	0	0	6	23
View_B_OSL_run_6	0	1	0	0	1	6	23
View_B_OSL_run_7	0	0	0	0	0	6	23
View_B_OSL_run_8	0	0	0	0	0	6	23
View_B_OSL_run_9	0	0	0	0	0	6	23
View_B_OSL_run_10	0	0	0	0	0	8	23
Average	0	0,1	0	0	0,1	6,3	23
Standard Deviation (SD)	0	0,3	0	0	0,3	0,640312424	0
View_B_VAPU_run_1	0	0	0	0	0	51	23
View_B_VAPU_run_2	0	0	0	0	0	58	23
View_B_VAPU_run_3	0	0	0	0	0	53	30
View_B_VAPU_run_4	0	0	1	0	1	57	24
View_B_VAPU_run_5	0	0	0	0	0	57	23
View_B_VAPU_run_6	0	0	0	0	0	56	25
View_B_VAPU_run_7	0	0	0	0	0	55	29
View_B_VAPU_run_8	0	0	0	0	0	58	28
View_B_VAPU_run_9	0	0	0	0	0	57	23
View_B_VAPU_run_10	0	0	0	0	0	64	23
Average	0	0	0,1	0	0,1	56,6	25,1
Standard Deviation (SD)	0	0	0,3	0	0,3	3,261901286	2,662705391
View_C_OSL_run_1	0	1	0	0	1	11	56
View_C_OSL_run_2	0	1	0	0	1	10	56
View_C_OSL_run_3	0	1	0	0	1	10	56
View_C_OSL_run_4	0	1	0	0	1	10	56
View_C_OSL_run_5	0	1	0	0	1	10	60
View_C_OSL_run_6	0	1	0	0	1	10	56
View_C_OSL_run_7	0	1	0	0	1	10	56
View_C_OSL_run_8	0	1	0	0	1	11	56
View_C_OSL_run_9	0	1	0	0	1	10	56
View_C_OSL_run_10	0	1	0	0	1	11	56
Average	0	1	0	0	1	10,3	56,4
Standard Deviation (SD)	0	0	0	0	0	0,458257569	1,2
View_C_VAPU_run_1	0	0	0	0	0	84	60
View_C_VAPU_run_2	0	1	0	0	1	90	60
View_C_VAPU_run_3	0	1	0	0	1	90	60

View_C_VAPU_run_4	0	0	0	0	0	78	61
View_C_VAPU_run_5	0	0	0	0	0	72	56
View_C_VAPU_run_6	0	1	0	0	1	76	60
View_C_VAPU_run_7	0	0	0	0	0	70	56
View_C_VAPU_run_8	0	0	0	0	0	86	60
View_C_VAPU_run_9	0	1	0	0	1	83	60
View_C_VAPU_run_10	0	1	0	0	1	86	56
Average	0	0,5	0	0	0,5	81,5	58,9
Standard Deviation (SD)	0	0,5	0	0	0,5	6,771262807	1,920937271
View_D_ZSL_run_1	0	0	0	0	0	20	168
View_D_ZSL_run_2	0	0	0	0	0	20	163
View_D_ZSL_run_3	0	0	0	0	0	21	163
View_D_ZSL_run_4	0	0	0	0	0	21	168
View_D_ZSL_run_5	0	0	0	0	0	21	168
View_D_ZSL_run_6	0	0	0	0	0	21	167
View_D_ZSL_run_7	0	0	0	0	0	21	168
View_D_ZSL_run_8	0	0	0	0	0	21	168
View_D_ZSL_run_9	0	0	0	0	0	20	168
View_D_ZSL_run_10	0	0	0	0	0	20	168
Average	0	0	0	0	0	20,6	166,9
Standard Deviation (SD)	0	0	0	0	0	0,489897949	1,972308292
View_D_VAPU_run_1	0	0	0	0	0	87	168
View_D_VAPU_run_2	0	1	1	1	3	81	172
View_D_VAPU_run_3	0	0	0	0	0	61	164
View_D_VAPU_run_4	0	1	1	0	2	89	201
View_D_VAPU_run_5	0	1	0	0	1	63	173
View_D_VAPU_run_6	0	1	0	0	1	118	167
View_D_VAPU_run_7	0	1	0	0	1	61	166
View_D_VAPU_run_8	0	2	0	0	2	120	165
View_D_VAPU_run_9	0	0	0	0	0	204	164
View_D_VAPU_run_10	0	1	0	0	1	63	167
Average	0	0,8	0,2	0,1	1,1	94,7	170,7
Standard Deviation (SD)	0	0,6	0,4	0,3	0,94339811	42,05960057	10,50761629

View file E evaluation process (Model: DeepSeek-V3, Temperature = 1)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	1	0	0	1	32	79
View_E_ZSL_run_2	1	0	0	1	35	43
View_E_ZSL_run_3	1	0	0	1	78	98
View_E_ZSL_run_4	1	0	0	1	36	41
View_E_ZSL_run_5	1	0	0	1	41	36
View_E_ZSL_run_6	1	0	0	1	58	47
View_E_ZSL_run_7	0	0	0	0	62	61
View_E_ZSL_run_8	1	0	0	1	71	65
View_E_ZSL_run_9	1	0	0	1	54	55
View_E_ZSL_run_10	1	0	0	1	75	70
Average	0,9	0	0	0,9	54,2	59,5
View_E_VAPU_run_1	0	0	0	0	48	125
View_E_VAPU_run_2	1	0	0	1	82	134
View_E_VAPU_run_3	1	0	0	1	67	209
View_E_VAPU_run_4	1	0	0	1	36	218
View_E_VAPU_run_5	0	1	0	1	68	172
View_E_VAPU_run_6	0	1	0	1	46	138
View_E_VAPU_run_7	1	0	0	1	66	136
View_E_VAPU_run_8	0	0	0	0	71	182
View_E_VAPU_run_9	1	1	0	2	78	248
View_E_VAPU_run_10	1	1	0	2	46	120
average	0,6	0,4	0	1	60,8	168,2

View file F evaluation process (Model: DeepSeek-V3, Temperature = 1)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC	Time
View_F_ZSL_run_1	0	0	0	0	4	36	5
View_F_ZSL_run_2	0	0	0	0	4	40	6
View_F_ZSL_run_3	0	0	0	0	4	35	6
View_F_ZSL_run_4	0	0	0	0	3	42	6
View_F_ZSL_run_5	0	0	0	0	4	35	6
View_F_ZSL_run_6	0	0	0	0	4	35	6
View_F_ZSL_run_7	0	0	0	0	4	40	7
View_F_ZSL_run_8	0	0	0	0	4	36	6
View_F_ZSL_run_9	0	0	0	0	4	35	5
View_F_ZSL_run_10	0	0	0	0	3	46	7
Avarage	0	0	0	0	3,8	38	6
View_F_VAPU_run_1	1	1	0	2	2	65	76
View_F_VAPU_run_2	0	0	0	0	2	73	83
View_F_VAPU_run_3	1	1	0	2	2	100	102
View_F_VAPU_run_4	0	0	0	0	1	84	97

View_F_VAPU_run_5	0	0	0	0	0	129	96
View_F_VAPU_run_6	1	1	0	2	0	108	101
View_F_VAPU_run_7	0	0	0	0	2	79	106
View_F_VAPU_run_8	0	0	0	0	0	38	90
View_F_VAPU_run_9	1	1	0	0	3	58	118
View_F_VAPU_run_10	1	1	1	3	1	86	87
Avarage	0,5	0,5	0,1	0,9	1,3	82	95,6

View files A, B, C and D evaluation process (Model: DeepSeek-V3, Temperature = 0)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	21	35
View_A_ZSL_run_2	0	0	0	0	0	19	35
View_A_ZSL_run_3	0	0	0	0	0	19	35
View_A_ZSL_run_4	0	0	0	0	0	20	35
View_A_ZSL_run_5	0	0	0	0	0	21	35
View_A_ZSL_run_6	0	0	0	0	0	19	35
View_A_ZSL_run_7	0	0	0	0	0	19	35
View_A_ZSL_run_8	0	0	0	0	0	19	35
View_A_ZSL_run_9	0	0	0	0	0	20	35
View_A_ZSL_run_10	0	0	0	0	0	20	35
Average	0	0	0	0	0	19,7	35
Standard Deviation (SD)	0	0	0	0	0	0,781024968	0
View_A_VAPU_run_1	0	0	0	0	0	51	35
View_A_VAPU_run_2	0	0	0	0	0	47	35
View_A_VAPU_run_3	0	0	0	0	0	47	35
View_A_VAPU_run_4	0	0	0	0	0	52	35
View_A_VAPU_run_5	0	0	0	0	0	73	35
View_A_VAPU_run_6	0	0	0	0	0	48	35
View_A_VAPU_run_7	0	0	0	0	0	49	35
View_A_VAPU_run_8	0	0	0	0	0	50	35
View_A_VAPU_run_9	0	0	0	0	0	53	35
View_A_VAPU_run_10	0	0	0	0	0	55	35
Average	0	0	0	0	0	52,5	35
Standard Deviation (SD)	0	0	0	0	0	7,269800547	0
View_B_OSL_run_1	0	0	0	0	0	44	23
View_B_OSL_run_2	0	0	0	0	0	35	23
View_B_OSL_run_3	0	0	0	0	0	43	23
View_B_OSL_run_4	0	0	0	0	0	33	23
View_B_OSL_run_5	0	0	0	0	0	33	23
View_B_OSL_run_6	0	0	0	0	0	36	23
View_B_OSL_run_7	0	0	0	0	0	35	23
View_B_OSL_run_8	0	0	0	0	0	21	23
View_B_OSL_run_9	0	0	0	0	0	32	23
View_B_OSL_run_10	0	0	0	0	0	34	23
Average	0	0	0	0	0	34,6	23
Standard Deviation (SD)	0	0	0	0	0	5,986651819	0
View_B_VAPU_run_1	0	1	0	0	1	145	24
View_B_VAPU_run_2	0	0	0	0	0	195	23
View_B_VAPU_run_3	0	0	0	0	0	135	24
View_B_VAPU_run_4	0	0	0	0	0	131	23
View_B_VAPU_run_5	0	0	0	0	0	134	23
View_B_VAPU_run_6	0	0	0	0	0	166	23
View_B_VAPU_run_7	0	0	0	0	0	178	23
View_B_VAPU_run_8	0	0	0	0	0	198	23
View_B_VAPU_run_9	0	0	0	0	0	147	23
View_B_VAPU_run_10	0	0	0	0	0	148	23
Average	0	0,1	0	0	0,1	157,7	23,2
Standard Deviation (SD)	0	0,3	0	0	0,3	23,78255663	0,4
View_C_OSL_run_1	0	1	0	0	1	69	56
View_C_OSL_run_2	0	1	0	0	1	43	56
View_C_OSL_run_3	0	1	0	0	1	54	56
View_C_OSL_run_4	0	1	0	0	1	58	56
View_C_OSL_run_5	0	1	0	0	1	50	56
View_C_OSL_run_6	0	1	0	0	1	37	56
View_C_OSL_run_7	0	1	0	0	1	38	56
View_C_OSL_run_8	0	1	0	0	1	46	56
View_C_OSL_run_9	0	1	0	0	1	41	56
View_C_OSL_run_10	0	1	0	0	1	45	56
Average	0	1	0	0	1	48,1	56
Standard Deviation (SD)	0	0	0	0	0	9,428149341	0
View_C_VAPU_run_1	0	0	0	0	0	145	56
View_C_VAPU_run_2	0	1	0	0	1	165	69
View_C_VAPU_run_3	0	0	0	0	0	145	60
View_C_VAPU_run_4	0	0	0	0	0	161	67
View_C_VAPU_run_5	0	0	0	0	0	109	56
View_C_VAPU_run_6	0	0	0	0	0	143	56

View_C_VAPU_run_7	0	0	0	0	0	148	71
View_C_VAPU_run_8	0	0	0	0	0	133	56
View_C_VAPU_run_9	0	0	0	0	0	131	60
View_C_VAPU_run_10	0	0	0	0	0	135	60
Average	0	0,1	0	0	0,1	141,5	61,1
Standard Deviation (SD)	0	0,3	0	0	0,3	15,10794493	5,503635162
View_D_ZSL_run_1	0	0	0	0	0	58	164
View_D_ZSL_run_2	0	0	0	0	0	59	162
View_D_ZSL_run_3	0	0	0	0	0	64	159
View_D_ZSL_run_4	0	0	0	0	0	68	157
View_D_ZSL_run_5	0	0	0	0	0	52	162
View_D_ZSL_run_6	0	0	0	0	0	58	162
View_D_ZSL_run_7	0	0	0	0	0	59	161
View_D_ZSL_run_8	0	0	0	0	0	72	162
View_D_ZSL_run_9	0	0	0	0	0	60	158
View_D_ZSL_run_10	0	0	0	0	0	65	162
Average	0	0	0	0	0	61,5	160,9
Standard Deviation (SD)	0	0	0	0	0	5,481788029	2,071231518
View_D_VAPU_run_1	0	0	0	0	0	116	163
View_D_VAPU_run_2	0	0	0	0	0	182	163
View_D_VAPU_run_3	0	1	0	0	1	236	155
View_D_VAPU_run_4	0	0	0	0	0	182	140
View_D_VAPU_run_5	0	0	0	0	0	188	164
View_D_VAPU_run_6	0	0	0	1	1	230	164
View_D_VAPU_run_7	0	0	0	0	0	150	153
View_D_VAPU_run_8	0	0	0	0	0	101	166
View_D_VAPU_run_9	0	0	0	0	0	146	162
View_D_VAPU_run_10	0	0	0	0	0	99	163
Average	0	0,1	0	0,1	0,2	163	159,3
Standard Deviation (SD)	0	0,3	0	0,3	0,4	46,66047578	7,537240874

View file E evaluation process (Model: DeepSeek-V3, Temperature = 0)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	1	0	0	1	59	58
View_E_ZSL_run_2	1	0	0	1	56	62
View_E_ZSL_run_3	1	0	0	1	63	52
View_E_ZSL_run_4	1	0	0	1	63	55
View_E_ZSL_run_5	1	0	0	1	61	68
View_E_ZSL_run_6	1	0	0	1	67	55
View_E_ZSL_run_7	1	0	0	1	57	51
View_E_ZSL_run_8	1	0	0	1	54	51
View_E_ZSL_run_9	1	0	0	1	63	53
View_E_ZSL_run_10	1	0	0	1	61	73
Average	1	0	0	1	60,4	57,8
View_E_VAPU_run_1	1	0	0	1	85	153
View_E_VAPU_run_2	0	0	0	0	69	284
View_E_VAPU_run_3	1	1	1	3	69	204
View_E_VAPU_run_4	0	0	0	0	41	247
View_E_VAPU_run_5	1	0	0	1	74	199
View_E_VAPU_run_6	1	0	0	1	70	264
View_E_VAPU_run_7	1	0	0	1	62	242
View_E_VAPU_run_8	1	0	0	1	51	167
View_E_VAPU_run_9	1	1	0	2	66	229
View_E_VAPU_run_10	1	0	0	1	45	114
average	0,8	0,2	0,1	1,1	63,2	210,3

View file F evaluation process (Model: DeepSeek-V3, Temperature = 0)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC	Time
View_F_ZSL_run_1	0	0	0	0	4	38	36
View_F_ZSL_run_2	0	0	0	0	3	44	41
View_F_ZSL_run_3	0	0	0	0	4	38	36
View_F_ZSL_run_4	0	0	0	0	4	38	44
View_F_ZSL_run_5	0	0	0	0	3	44	39
View_F_ZSL_run_6	0	0	0	0	4	38	37
View_F_ZSL_run_7	0	0	0	0	3	44	43
View_F_ZSL_run_8	0	0	0	0	4	35	34
View_F_ZSL_run_9	0	0	0	0	4	38	45
View_F_ZSL_run_10	0	0	0	0	0	44	45
Avarage	0	0	0	0	3,3	40,1	40
View_F_VAPU_run_1	1	1	1	3	2	77	229
View_F_VAPU_run_2	1	1	1	3	2	76	190
View_F_VAPU_run_3	0	0	0	0	4	39	124
View_F_VAPU_run_4	0	0	0	0	4	36	160
View_F_VAPU_run_5	1	1	0	2	4	35	141
View_F_VAPU_run_6	1	1	0	2	3	58	273
View_F_VAPU_run_7	0	0	0	0	2	76	261

View_F_VAPU_run_8	1	1	1	3	2	72	237
View_F_VAPU_run_9	0	0	0	0	2	72	287
View_F_VAPU_run_10	0	0	0	0	4	40	167
Avarage	0,5	0,5	0,3	1,3	2,9	58,1	206,9

View files A, B, C and D evaluation process (Model: Claude Sonnet 3.5, Temperature = 1)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	6	41
View_A_ZSL_run_2	0	0	0	0	0	6	41
View_A_ZSL_run_3	0	0	0	0	0	7	39
View_A_ZSL_run_4	0	0	0	0	0	6	34
View_A_ZSL_run_5	1	0	0	0	1	7	41
View_A_ZSL_run_6	1	0	0	0	1	7	41
View_A_ZSL_run_7	0	0	0	0	0	7	41
View_A_ZSL_run_8	1	0	0	0	1	7	41
View_A_ZSL_run_9	1	0	0	0	1	6	41
View_A_ZSL_run_10	0	0	0	0	0	7	41
Average	0,4	0	0	0	0,4	6,6	40,1
Standard Deviation (SD)	0,489897949	0	0	0	0,48989795	0,489897949	2,11896201
View_A_VAPU_run_1	1	0	0	0	1	63	43
View_A_VAPU_run_2	1	0	0	0	1	40	62
View_A_VAPU_run_3	1	0	1	0	2	50	50
View_A_VAPU_run_4	1	0	1	0	2	76	50
View_A_VAPU_run_5	1	0	0	0	1	37	46
View_A_VAPU_run_6	1	0	0	0	1	35	44
View_A_VAPU_run_7	1	0	0	0	1	57	47
View_A_VAPU_run_8	1	0	0	0	1	50	44
View_A_VAPU_run_9	1	0	0	0	1	37	46
View_A_VAPU_run_10	1	0	0	0	1	32	37
Average	1	0	0,2	0	1,2	47,7	46,9
Standard Deviation (SD)	0	0	0,4	0	0,4	13,52072483	6,155485359
View_B_OSL_run_1	0	0	0	0	0	8	26
View_B_OSL_run_2	0	0	0	0	0	8	25
View_B_OSL_run_3	0	0	0	0	0	8	25
View_B_OSL_run_4	0	0	0	0	0	5	25
View_B_OSL_run_5	0	0	0	0	0	8	26
View_B_OSL_run_6	0	0	0	0	0	6	26
View_B_OSL_run_7	0	0	0	0	0	6	25
View_B_OSL_run_8	0	0	0	0	0	5	28
View_B_OSL_run_9	0	0	0	0	0	9	24
View_B_OSL_run_10	0	0	0	0	0	8	26
Average	0	0	0	0	0	7,1	25,6
Standard Deviation (SD)	0	0	0	0	0	1,374772708	1,019803903
View_B_VAPU_run_1	0	0	0	0	0	76	24
View_B_VAPU_run_2	0	0	0	0	0	90	34
View_B_VAPU_run_3	0	0	0	0	0	96	43
View_B_VAPU_run_4	1	0	0	0	1	68	28
View_B_VAPU_run_5	0	0	0	0	0	75	38
View_B_VAPU_run_6	1	0	0	0	1	82	43
View_B_VAPU_run_7	0	0	0	0	0	88	37
View_B_VAPU_run_8	0	0	0	0	0	89	40
View_B_VAPU_run_9	0	0	0	0	0	92	50
View_B_VAPU_run_10	0	0	0	0	0	96	51
Average	0,2	0	0	0	0,2	85,2	38,8
Standard Deviation (SD)	0,4	0	0	0	0,4	9,053176238	8,207313811
View_C_OSL_run_1	0	0	0	0	0	9	60
View_C_OSL_run_2	0	0	0	0	0	14	60
View_C_OSL_run_3	0	0	0	0	0	14	65
View_C_OSL_run_4	0	0	0	0	0	10	73
View_C_OSL_run_5	0	0	0	0	0	11	77
View_C_OSL_run_6	0	0	0	0	0	12	77
View_C_OSL_run_7	0	0	0	0	0	11	60
View_C_OSL_run_8	0	0	0	0	0	13	77
View_C_OSL_run_9	0	0	0	0	0	10	66
View_C_OSL_run_10	0	0	0	0	0	15	75
Average	0	0	0	0	0	11,9	69
Standard Deviation (SD)	0	0	0	0	0	7,155417528	1,920937271
View_C_VAPU_run_1	0	0	0	0	0	90	74
View_C_VAPU_run_2	0	0	0	0	0	89	72
View_C_VAPU_run_3	1	2	0	0	3	90	76
View_C_VAPU_run_4	0	0	0	0	0	76	76
View_C_VAPU_run_5	1	0	0	0	1	80	75
View_C_VAPU_run_6	0	0	0	0	0	76	75
View_C_VAPU_run_7	0	0	0	0	0	74	76
View_C_VAPU_run_8	0	0	0	0	0	76	72
View_C_VAPU_run_9	0	0	0	0	0	95	76

View_C_VAPU_run_10	0	0	0	0	0	71	77
Average	0,2	0,2	0	0	0,4	81,7	74,9
Standard Deviation (SD)	0,4	0,6	0	0	0,91651514	8,013114251	1,640121947
View_D_ZSL_run_1	1	1	0	0	2	21	148
View_D_ZSL_run_2	0	1	0	0	1	21	156
View_D_ZSL_run_3	0	1	0	0	1	18	170
View_D_ZSL_run_4	0	1	0	0	1	21	172
View_D_ZSL_run_5	0	1	0	0	1	38	193
View_D_ZSL_run_6	0	1	0	0	1	20	170
View_D_ZSL_run_7	0	0	0	0	0	20	177
View_D_ZSL_run_8	0	1	0	0	1	38	166
View_D_ZSL_run_9	0	1	0	0	1	20	159
View_D_ZSL_run_10	0	1	0	0	1	20	190
Average	0,1	0,9	0	0	1	23,7	170,1
Standard Deviation (SD)	0,3	0,3	0	0	0,4472136	13,41230778	7,197916365
View_D_VAPU_run_1	0	1	0	0	1	64	181
View_D_VAPU_run_2	0	0	0	0	0	99	173
View_D_VAPU_run_3	0	2	0	0	2	91	120
View_D_VAPU_run_4	1	2	0	0	3	88	113
View_D_VAPU_run_5	1	1	0	0	2	98	158
View_D_VAPU_run_6	1	1	0	0	2	128	199
View_D_VAPU_run_7	0	0	0	0	0	86	170
View_D_VAPU_run_8	1	1	0	0	2	81	157
View_D_VAPU_run_9	1	0	1	0	2	136	216
View_D_VAPU_run_10	0	1	0	0	1	67	174
Average	0,5	0,9	0,1	0	1,5	93,8	166,1
Standard Deviation (SD)	0,5	0,7	0,3	0	0,92195445	22,0626381	30,0214923

View file E evaluation process (Model: Claude Sonnet 3.5, Temperature = 1)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	1	1	1	3	78	9
View_E_ZSL_run_2	1	1	0	2	49	8
View_E_ZSL_run_3	0	1	0	1	76	9
View_E_ZSL_run_4	0	0	0	0	97	13
View_E_ZSL_run_5	0	0	0	0	63	10
View_E_ZSL_run_6	1	0	0	1	59	9
View_E_ZSL_run_7	1	1	1	3	85	9
View_E_ZSL_run_8	0	0	0	0	86	11
View_E_ZSL_run_9	1	0	0	1	51	11
View_E_ZSL_run_10	0	0	0	0	54	7
Average	0,5	0,4	0,2	1,1	69,8	9,6
View_E_VAPU_run_1	0	0	0	0	209	132
View_E_VAPU_run_2	0	0	0	0	166	129
View_E_VAPU_run_3	0	0	0	0	116	117
View_E_VAPU_run_4	0	0	0	0	136	102
View_E_VAPU_run_5	0	0	0	0	120	119
View_E_VAPU_run_6	0	0	0	0	177	138
View_E_VAPU_run_7	0	0	0	0	154	130
View_E_VAPU_run_8	0	0	0	0	129	90
View_E_VAPU_run_9	0	0	0	0	129	120
View_E_VAPU_run_10	0	0	0	0	161	111
average	0	0	0	0	149,7	118,8

View file F evaluation process (Model: Claude Sonnet 3.5, Temperature = 1)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC	Time
View_F_ZSL_run_1	0	0	0	0	3	44	6
View_F_ZSL_run_2	1	1	0	2	4	32	5
View_F_ZSL_run_3	1	1	0	2	4	32	5
View_F_ZSL_run_4	1	1	0	2	4	32	5
View_F_ZSL_run_5	0	0	0	0	3	44	6
View_F_ZSL_run_6	1	1	0	2	4	31	8
View_F_ZSL_run_7	1	1	0	2	4	31	6
View_F_ZSL_run_8	1	1	0	2	4	32	5
View_F_ZSL_run_9	1	1	0	2	4	32	5
View_F_ZSL_run_10	1	1	0	2	4	34	5
Avarage	0,8	0,8	0	1,6	3,8	34,4	5,6
View_F_VAPU_run_1	0	0	0	0	0	142	110
View_F_VAPU_run_2	0	0	0	0	1	98	53
View_F_VAPU_run_3	0	0	0	0	4	50	56
View_F_VAPU_run_4	0	0	0	0	0	179	118
View_F_VAPU_run_5	0	0	0	0	0	141	99
View_F_VAPU_run_6	0	0	0	0	0	101	118
View_F_VAPU_run_7	0	0	0	0	0	82	94
View_F_VAPU_run_8	1	1	1	3	1	102	108
View_F_VAPU_run_9	0	0	0	0	0	107	91
View_F_VAPU_run_10	1	1	1	3	0	129	84

Avarage	0,2	0,2	0,2	0,6	0,6	113,1	93,1

View files A, B, C and D evaluation process (Model: Claude Sonnet 3.5, Temperature = 0)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	7	41
View_A_ZSL_run_2	1	0	0	0	1	6	41
View_A_ZSL_run_3	1	0	0	0	1	7	41
View_A_ZSL_run_4	1	0	0	0	1	7	41
View_A_ZSL_run_5	0	0	0	0	0	8	41
View_A_ZSL_run_6	1	0	0	0	1	6	41
View_A_ZSL_run_7	0	0	0	0	0	7	41
View_A_ZSL_run_8	0	0	0	0	0	7	41
View_A_ZSL_run_9	0	0	0	0	0	6	41
View_A_ZSL_run_10	0	0	0	0	0	7	41
Average	0,4	0	0	0	0,4	6,8	41
Standard Deviation (SD)	0,489897949	0	0	0	0,48989795	0,6	0
View_A_VAPU_run_1	1	0	0	0	1	53	46
View_A_VAPU_run_2	1	0	0	0	1	47	43
View_A_VAPU_run_3	1	0	0	0	1	34	43
View_A_VAPU_run_4	1	0	0	0	1	37	43
View_A_VAPU_run_5	1	0	0	0	1	40	44
View_A_VAPU_run_6	1	0	0	0	1	43	48
View_A_VAPU_run_7	1	0	0	0	1	37	43
View_A_VAPU_run_8	1	0	0	0	1	35	43
View_A_VAPU_run_9	1	0	0	0	1	42	43
View_A_VAPU_run_10	1	0	0	0	1	50	42
Average	1	0	0	0	1	41,8	43,8
Standard Deviation (SD)	0	0	0	0	0	6,144916598	1,720465053
View_B_OSL_run_1	0	0	0	0	0	5	23
View_B_OSL_run_2	0	0	0	0	0	5	23
View_B_OSL_run_3	0	0	0	0	0	7	23
View_B_OSL_run_4	0	0	0	0	0	5	23
View_B_OSL_run_5	0	0	0	0	0	5	23
View_B_OSL_run_6	0	0	0	0	0	5	24
View_B_OSL_run_7	0	0	0	0	0	5	24
View_B_OSL_run_8	0	0	0	0	0	5	23
View_B_OSL_run_9	0	0	0	0	0	5	24
View_B_OSL_run_10	0	0	0	0	0	5	23
Average	0	0	0	0	0	5,2	23,3
Standard Deviation (SD)	0	0	0	0	0	0,6	0,458257569
View_B_VAPU_run_1	0	0	1	0	1	215	43
View_B_VAPU_run_2	0	0	0	0	0	51	26
View_B_VAPU_run_3	0	0	0	0	0	55	33
View_B_VAPU_run_4	0	0	0	0	0	70	29
View_B_VAPU_run_5	0	0	0	0	0	54	33
View_B_VAPU_run_6	0	0	0	0	0	72	37
View_B_VAPU_run_7	0	0	0	0	0	56	29
View_B_VAPU_run_8	0	0	0	0	0	60	43
View_B_VAPU_run_9	0	0	0	0	0	61	43
View_B_VAPU_run_10	1	0	0	0	1	73	53
Average	0,1	0	0,1	0	0,2	76,7	36,9
Standard Deviation (SD)	0,3	0	0,3	0	0,4	46,69914346	8,030566605
View_C_OSL_run_1	0	0	0	0	0	10	75
View_C_OSL_run_2	0	0	0	0	0	9	81
View_C_OSL_run_3	0	0	0	0	0	9	71
View_C_OSL_run_4	0	0	0	0	0	9	81
View_C_OSL_run_5	0	0	0	0	0	9	79
View_C_OSL_run_6	0	0	0	0	0	9	71
View_C_OSL_run_7	0	0	0	0	0	10	81
View_C_OSL_run_8	0	0	0	0	0	9	81
View_C_OSL_run_9	0	0	0	0	0	10	81
View_C_OSL_run_10	0	0	0	0	0	9	81
Average	0	0	0	0	0	9,3	78,2
Standard Deviation (SD)	0	0	0	0	0	0,458257569	4,019950248
View_C_VAPU_run_1	0	0	0	0	0	69	76
View_C_VAPU_run_2	0	0	0	0	0	85	76
View_C_VAPU_run_3	0	0	0	0	0	89	78
View_C_VAPU_run_4	0	0	0	0	0	75	76
View_C_VAPU_run_5	0	0	0	0	0	75	74
View_C_VAPU_run_6	0	0	0	0	0	82	74
View_C_VAPU_run_7	0	0	0	0	0	85	75
View_C_VAPU_run_8	0	0	0	0	0	87	76
View_C_VAPU_run_9	0	0	0	0	0	75	80
View_C_VAPU_run_10	0	0	0	0	0	83	74
Average	0	0	0	0	0	80,5	75,9
Standard Deviation (SD)	0	0	0	0	0	6,216912417	1,813835715

View_D_ZSL_run_1	0	1	0	0	1	19	182
View_D_ZSL_run_2	0	1	0	0	1	19	182
View_D_ZSL_run_3	0	1	0	0	1	20	182
View_D_ZSL_run_4	0	1	0	0	1	20	182
View_D_ZSL_run_5	0	1	0	0	1	27	182
View_D_ZSL_run_6	0	1	0	0	1	24	182
View_D_ZSL_run_7	0	1	0	0	1	21	182
View_D_ZSL_run_8	0	1	0	0	1	22	182
View_D_ZSL_run_9	0	1	0	0	1	23	182
View_D_ZSL_run_10	0	1	0	0	1	25	182
Average	0	1	0	0	1	22	182
Standard Deviation (SD)	0	0	0	0	0	2,569046516	0
View_D_VAPU_run_1	1	0	0	0	1	93	199
View_D_VAPU_run_2	0	1	0	0	1	94	178
View_D_VAPU_run_3	0	1	0	0	1	107	179
View_D_VAPU_run_4	0	0	0	0	0	105	145
View_D_VAPU_run_5	0	0	0	0	0	90	193
View_D_VAPU_run_6	0	1	0	0	1	66	184
View_D_VAPU_run_7	1	1	0	0	2	79	195
View_D_VAPU_run_8	1	0	0	0	1	94	199
View_D_VAPU_run_9	1	0	0	0	1	90	190
View_D_VAPU_run_10	1	1	0	0	2	89	172
Average	0,5	0,5	0	0	1	90,7	183,4
Standard Deviation (SD)	0,5	0,5	0	0	0,63245553	11,1718396	15,52546296

View file E evaluation process (Model: Claude Sonnet 3.5, Temperature = 0)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	1	1	1	3	82	13
View_E_ZSL_run_2	1	1	0	3	55	7
View_E_ZSL_run_3	1	1	0	2	55	7
View_E_ZSL_run_4	1	1	0	2	68	9
View_E_ZSL_run_5	1	1	0	2	61	10
View_E_ZSL_run_6	1	1	0	2	65	11
View_E_ZSL_run_7	1	1	0	2	79	12
View_E_ZSL_run_8	1	1	1	3	80	9
View_E_ZSL_run_9	1	1	0	2	47	7
View_E_ZSL_run_10	1	1	0	2	55	7
Average	1	1	0,2	2,3	64,7	9,2
View_E_VAPU_run_1	0	0	0	0	125	113
View_E_VAPU_run_2	0	0	0	0	126	114
View_E_VAPU_run_3	0	0	0	0	146	102
View_E_VAPU_run_4	0	0	0	0	178	111
View_E_VAPU_run_5	0	0	0	0	200	123
View_E_VAPU_run_6	0	0	0	0	162	120
View_E_VAPU_run_7	0	0	0	0	33	91
View_E_VAPU_run_8	0	0	0	0	134	91
View_E_VAPU_run_9	0	0	0	0	162	105
View_E_VAPU_run_10	0	1	0	1	140	97
average	0	0,1	0	0,1	140,6	106,7

View file F evaluation process (Model: Claude Sonnet 3.5, Temperature = 0)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC	Time
View_F_ZSL_run_1	1	1	1	3	4	32	23
View_F_ZSL_run_2	1	1	1	3	4	32	5
View_F_ZSL_run_3	1	1	1	3	4	32	5
View_F_ZSL_run_4	1	1	1	3	4	32	5
View_F_ZSL_run_5	1	1	1	3	4	31	4
View_F_ZSL_run_6	1	1	1	3	4	31	5
View_F_ZSL_run_7	1	1	1	3	4	32	4
View_F_ZSL_run_8	1	1	1	3	4	32	4
View_F_ZSL_run_9	1	1	1	3	4	32	4
View_F_ZSL_run_10	1	1	1	3	4	32	4
Avarage	1	1	1	3	4	31,8	6,3
View_F_VAPU_run_1	0	0	0	0	0	102	299
View_F_VAPU_run_2	0	0	0	0	2	93	66
View_F_VAPU_run_3	0	0	0	0	0	130	140
View_F_VAPU_run_4	1	1	1	3	0	133	114
View_F_VAPU_run_5	1	1	0	2	2	42	149
View_F_VAPU_run_6	0	0	0	0	2	52	88
View_F_VAPU_run_7	1	1	0	2	2	96	84
View_F_VAPU_run_8	0	0	0	0	3	70	85
View_F_VAPU_run_9	0	0	0	0	4	51	92
View_F_VAPU_run_10	1	1	0	2	1	103	127
Avarage	0,4	0,4	0,1	0,9	1,6	87,2	124,4

View files A, B, C and D evaluation process (Model: Nova Pro 1.0, Temperature = 1)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)
View_A_ZSL_run_1	0	0	0	0	0	6	36
View_A_ZSL_run_2	0	0	0	0	0	5	35
View_A_ZSL_run_3	0	0	0	0	0	5	35
View_A_ZSL_run_4	0	0	0	0	0	6	35
View_A_ZSL_run_5	0	0	0	0	0	5	35
View_A_ZSL_run_6	0	0	0	0	0	5	35
View_A_ZSL_run_7	0	0	0	0	0	5	35
View_A_ZSL_run_8	0	0	0	0	0	5	35
View_A_ZSL_run_9	0	0	0	0	0	5	34
View_A_ZSL_run_10	1	0	0	0	1	5	35
Average	0,1	0	0	0	0,1	5,2	35
Standard Deviation (SD)	0,3	0	0	0	0,3	0,4	0,447213595
View_A_VAPU_run_1	1	0	0	0	1	133	35
View_A_VAPU_run_2	1	0	0	0	1	43	35
View_A_VAPU_run_3	0	0	1	0	1	53	48
View_A_VAPU_run_4	1	0	0	0	1	108	60
View_A_VAPU_run_5	0	0	0	0	0	49	35
View_A_VAPU_run_6	1	1	0	0	2	55	35
View_A_VAPU_run_7	0	0	0	0	0	53	35
View_A_VAPU_run_8	0	0	1	0	1	56	39
View_A_VAPU_run_9	0	0	0	0	0	46	35
View_A_VAPU_run_10	1	0	0	0	1	48	49
Average	0,5	0,1	0,2	0	0,8	64,4	40,6
Standard Deviation (SD)	0,5	0,3	0,4	0	0,6	28,85896741	8,321057625
View_B_OSL_run_1	0	1	0	0	1	5	25
View_B_OSL_run_2	0	1	0	0	1	5	23
View_B_OSL_run_3	0	1	0	0	1	5	25
View_B_OSL_run_4	0	1	0	0	1	5	25
View_B_OSL_run_5	0	1	0	0	1	5	25
View_B_OSL_run_6	0	1	0	0	1	5	25
View_B_OSL_run_7	0	1	0	0	1	5	25
View_B_OSL_run_8	0	1	0	0	1	5	25
View_B_OSL_run_9	0	1	0	0	1	5	23
View_B_OSL_run_10	0	1	0	0	1	5	23
Average	0	1	0	0	1	5	24,4
Standard Deviation (SD)	0	0	0	0	0	0	0,916515139
View_B_VAPU_run_1	2	2	0	0	4	60	23
View_B_VAPU_run_2	1	0	0	0	1	75	23
View_B_VAPU_run_3	0	0	0	0	0	65	23
View_B_VAPU_run_4	1	0	0	0	1	119	33
View_B_VAPU_run_5	0	0	0	0	0	60	22
View_B_VAPU_run_6	0	0	1	0	1	57	30
View_B_VAPU_run_7	1	1	0	0	2	52	23
View_B_VAPU_run_8	1	0	0	0	1	90	21
View_B_VAPU_run_9	0	0	0	0	0	58	23
View_B_VAPU_run_10	1	0	0	0	1	113	27
Average	0,7	0,3	0,1	0	1,1	74,9	24,8
Standard Deviation (SD)	0,640312424	0,640312424	0,3	0	1,13578167	23,01499511	3,709447398
View_C_OSL_run_1	0	2	0	0	2	9	62
View_C_OSL_run_2	0	1	0	0	1	9	59
View_C_OSL_run_3	0	0	0	0	0	9	58
View_C_OSL_run_4	0	1	0	0	1	9	59
View_C_OSL_run_5	0	1	0	0	1	9	58
View_C_OSL_run_6	0	2	0	0	2	9	60
View_C_OSL_run_7	0	1	0	0	1	9	56
View_C_OSL_run_8	0	1	0	0	1	8	63
View_C_OSL_run_9	0	1	0	0	1	9	62
View_C_OSL_run_10	0	1	0	0	1	10	60
Average	0	1,1	0	0	1,1	9	59,7
Standard Deviation (SD)	0	0,538516481	0	0	0,53851648	0,447213595	2,051828453
View_C_VAPU_run_1	0	0	0	0	0	108	67
View_C_VAPU_run_2	1	0	0	0	1	103	82
View_C_VAPU_run_3	0	2	0	0	2	90	63
View_C_VAPU_run_4	0	0	0	0	0	97	75
View_C_VAPU_run_5	0	2	0	0	2	99	56
View_C_VAPU_run_6	0	1	0	0	1	72	67
View_C_VAPU_run_7	0	1	0	0	1	63	67
View_C_VAPU_run_8	0	1	0	0	1	56	70
View_C_VAPU_run_9	2	1	0	1	4	99	52
View_C_VAPU_run_10	0	2	0	0	2	80	62
Average	0,3	1	0	0,1	1,4	86,7	66,1
Standard Deviation (SD)	0,640312424	0,774596669	0	0,3	1,11355287	17,04142013	8,227393269
View_D_ZSL_run_1	0	0	0	0	0	21	172

View_D_ZSL_run_2	0	0	0	0	0	18	176
View_D_ZSL_run_3	0	2	0	0	2	21	174
View_D_ZSL_run_4	0	0	0	0	0	19	162
View_D_ZSL_run_5	0	1	0	0	1	19	179
View_D_ZSL_run_6	0	2	0	0	2	17	137
View_D_ZSL_run_7	0	0	0	0	0	19	157
View_D_ZSL_run_8	0	1	0	0	1	18	148
View_D_ZSL_run_9	0	1	0	0	1	18	147
View_D_ZSL_run_10	0	1	0	0	1	16	157
Average	0	0,8	0	0	0,8	160,9	18,6
Standard Deviation (SD)	0	0,748331477	0	0	0,74833148	13,4643975	1,496662955
View_D_VAPU_run_1	0	2	1	0	3	104	162
View_D_VAPU_run_2	0	0	0	0	0	179	161
View_D_VAPU_run_3	0	1	0	0	1	128	172
View_D_VAPU_run_4	0	0	0	0	0	136	183
View_D_VAPU_run_5	0	2	0	0	2	161	164
View_D_VAPU_run_6	0	2	0	0	2	120	172
View_D_VAPU_run_7	0	0	0	0	0	118	166
View_D_VAPU_run_8	0	0	0	0	0	151	173
View_D_VAPU_run_9	0	2	1	0	3	136	172
View_D_VAPU_run_10	0	0	0	0	0	166	158
Average	0	0,9	0,2	0	1,1	139,9	168,3
Standard Deviation (SD)	0	0,943398113	0,4	0	1,22065556	22,61614468	7,085901495

View file E evaluation process (Model: Nova Pro 1.0, Temperature = 1)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	1	0	0	1	43	6
View_E_ZSL_run_2	1	1	0	2	57	6
View_E_ZSL_run_3	1	0	0	1	35	4
View_E_ZSL_run_4	1	0	0	1	56	7
View_E_ZSL_run_5	1	0	0	1	44	7
View_E_ZSL_run_6	0	0	0	0	45	7
View_E_ZSL_run_7	0	0	0	0	39	7
View_E_ZSL_run_8	1	0	0	1	53	10
View_E_ZSL_run_9	0	0	0	0	45	6
View_E_ZSL_run_10	1	0	0	1	66	7
Average	0,7	0,1	0	0,8	48,3	6,7
View_E_VAPU_run_1	1	0	0	1	30	34
View_E_VAPU_run_2	1	0	0	1	53	58
View_E_VAPU_run_3	1	0	0	1	63	54
View_E_VAPU_run_4	0	0	0	0	37	46
View_E_VAPU_run_5	0	0	0	0	42	53
View_E_VAPU_run_6	0	0	0	0	43	49
View_E_VAPU_run_7	0	0	0	0	38	51
View_E_VAPU_run_8	0	0	0	0	47	38
View_E_VAPU_run_9	1	0	0	1	46	45
View_E_VAPU_run_10	1	0	0	1	53	56
average	0,5	0	0	0,5	45,2	48,4

View file F evaluation process (Model: Nova Pro 1.0, Temperature = 1)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC	Time
View_F_ZSL_run_1	0	0	0	0	4	36	5
View_F_ZSL_run_2	0	0	0	0	1	78	10
View_F_ZSL_run_3	0	0	0	0	4	42	5
View_F_ZSL_run_4	0	0	0	0	4	35	5
View_F_ZSL_run_5	1	1	0	2	4	38	5
View_F_ZSL_run_6	0	0	0	0	1	44	6
View_F_ZSL_run_7	0	0	0	0	4	34	5
View_F_ZSL_run_8	1	1	1	3	0	108	12
View_F_ZSL_run_9	0	0	0	0	3	47	6
View_F_ZSL_run_10	0	0	0	0	4	24	4
Avarage	0,2	0,2	0,1	0,5	2,9	48,6	6,3
View_F_VAPU_run_1	1	1	0	2	4	35	60
View_F_VAPU_run_2	0	0	0	0	3	73	82
View_F_VAPU_run_3	1	1	0	2	4	41	81
View_F_VAPU_run_4	1	1	0	2	3	39	75
View_F_VAPU_run_5	1	1	1	3	1	107	76
View_F_VAPU_run_6	0	0	0	0	3	67	75
View_F_VAPU_run_7	0	0	0	0	0	91	123
View_F_VAPU_run_8	0	0	0	0	3	52	64
View_F_VAPU_run_9	0	0	0	0	2	48	66
View_F_VAPU_run_10	0	0	0	0	4	39	69
Avarage	0,4	0,4	0,1	0,9	2,7	59,2	77,1

View files A, B, C and D evaluation process (Model: Nova Pro 1.0, Temperature = 0)

Run	Fatal errors	Runtime Errors	Content Errors	Missing additional content	Total Errors	Time (seconds)	Lines Of Code (LOC)

View_A_ZSL_run_1	0	0	0	0	0	5	35
View_A_ZSL_run_2	0	0	0	0	0	5	35
View_A_ZSL_run_3	0	0	0	0	0	5	35
View_A_ZSL_run_4	0	0	0	0	0	5	35
View_A_ZSL_run_5	0	0	0	0	0	5	35
View_A_ZSL_run_6	0	0	0	0	0	5	35
View_A_ZSL_run_7	0	0	0	0	0	5	35
View_A_ZSL_run_8	0	0	0	0	0	5	35
View_A_ZSL_run_9	0	0	0	0	0	5	35
View_A_ZSL_run_10	0	0	0	0	0	5	35
Average	0	0	0	0	0	5	35
Standard Deviation (SD)	0	0	0	0	0	0	0
View_A_VAPU_run_1	0	0	0	0	0	43	35
View_A_VAPU_run_2	0	1	0	0	1	44	37
View_A_VAPU_run_3	0	0	0	0	0	54	41
View_A_VAPU_run_4	0	0	0	0	0	60	50
View_A_VAPU_run_5	0	0	0	0	0	51	36
View_A_VAPU_run_6	0	0	0	0	0	51	35
View_A_VAPU_run_7	0	0	0	0	0	38	37
View_A_VAPU_run_8	0	0	0	0	0	47	37
View_A_VAPU_run_9	0	0	0	0	0	51	39
View_A_VAPU_run_10	0	0	0	0	0	62	35
Average	0	0,1	0	0	0,1	50,1	38,2
Standard Deviation (SD)	0	0,3	0	0	0,3	7,077428912	4,331281566
View_B_OSL_run_1	1	0	0	0	1	4	25
View_B_OSL_run_2	0	0	0	0	0	4	23
View_B_OSL_run_3	0	0	0	0	0	4	25
View_B_OSL_run_4	0	0	0	0	0	4	25
View_B_OSL_run_5	1	0	0	0	1	4	25
View_B_OSL_run_6	0	0	0	0	0	5	25
View_B_OSL_run_7	0	0	0	0	0	5	25
View_B_OSL_run_8	0	0	0	0	0	5	23
View_B_OSL_run_9	0	0	0	0	0	5	23
View_B_OSL_run_10	0	0	0	0	0	5	23
Average	0,2	0	0	0	0,2	4,5	24,2
Standard Deviation (SD)	0,4	0	0	0	0,4	0,5	0,979795897
View_B_VAPU_run_1	1	0	0	0	1	57	36
View_B_VAPU_run_2	2	0	0	0	2	54	23
View_B_VAPU_run_3	0	0	0	0	0	40	23
View_B_VAPU_run_4	1	0	0	0	1	45	23
View_B_VAPU_run_5	0	0	0	0	0	43	23
View_B_VAPU_run_6	0	0	0	0	0	47	25
View_B_VAPU_run_7	0	0	0	0	0	40	23
View_B_VAPU_run_8	1	0	0	0	1	51	23
View_B_VAPU_run_9	0	0	0	0	0	37	23
View_B_VAPU_run_10	0	0	0	0	0	53	23
Average	0,5	0	0	0	0,5	46,7	24,5
Standard Deviation (SD)	0,670820393	0	0	0	0,67082039	6,466065264	3,879432948
View_C_OSL_run_1	0	0	0	0	0	7	56
View_C_OSL_run_2	0	0	0	0	0	7	56
View_C_OSL_run_3	0	0	0	0	0	7	56
View_C_OSL_run_4	0	0	0	0	0	7	56
View_C_OSL_run_5	0	0	0	0	0	7	58
View_C_OSL_run_6	0	0	0	0	0	7	56
View_C_OSL_run_7	0	0	0	0	0	7	56
View_C_OSL_run_8	0	0	0	0	0	7	56
View_C_OSL_run_9	0	1	0	0	1	8	60
View_C_OSL_run_10	0	1	0	0	1	7	56
Average	0	0,2	0	0	0,2	7,1	56,6
Standard Deviation (SD)	0	0,4	0	0	0,4	0,3	1,280624847
View_C_VAPU_run_1	0	1	0	0	1	65	69
View_C_VAPU_run_2	0	0	0	0	0	75	67
View_C_VAPU_run_3	0	0	0	0	0	77	67
View_C_VAPU_run_4	0	0	0	0	0	65	67
View_C_VAPU_run_5	0	1	0	0	1	74	71
View_C_VAPU_run_6	0	0	0	0	0	61	56
View_C_VAPU_run_7	1	1	0	0	2	66	65
View_C_VAPU_run_8	0	0	0	0	0	55	69
View_C_VAPU_run_9	1	0	0	0	1	85	57
View_C_VAPU_run_10	0	1	0	0	1	69	75
Average	0,2	0,4	0	0	0,6	69,2	66,3
Standard Deviation (SD)	0,4	0,489897949	0	0	0,66332496	8,255906976	5,550675635
View_D_ZSL_run_1	0	0	0	0	0	13	184
View_D_ZSL_run_2	0	0	0	0	0	12	183
View_D_ZSL_run_3	0	0	0	0	0	13	183
View_D_ZSL_run_4	0	0	0	0	0	12	182

View_D_ZSL_run_5	0	0	0	0	0	11	184
View_D_ZSL_run_6	0	0	0	0	0	13	184
View_D_ZSL_run_7	0	0	0	0	0	12	184
View_D_ZSL_run_8	0	0	0	0	0	13	181
View_D_ZSL_run_9	0	0	0	0	0	12	184
View_D_ZSL_run_10	0	0	0	0	0	11	176
Average	0	0	0	0	0	12,2	182,5
Standard Deviation (SD)	0	0	0	0	0	0,748331477	2,376972865
View_D_VAPU_run_1	1	1	0	0	2	102	162
View_D_VAPU_run_2	0	0	0	0	0	95	182
View_D_VAPU_run_3	0	2	0	0	2	107	152
View_D_VAPU_run_4	0	0	0	0	0	100	181
View_D_VAPU_run_5	0	0	1	0	1	111	171
View_D_VAPU_run_6	0	1	0	0	1	92	157
View_D_VAPU_run_7	0	2	0	0	2	98	156
View_D_VAPU_run_8	1	0	0	0	1	87	187
View_D_VAPU_run_9	0	1	0	0	1	84	177
View_D_VAPU_run_10	0	1	0	0	1	91	156
Average	0,2	0,8	0,1	0	1,1	96,7	168,1
Standard Deviation (SD)	0,4	0,748331477	0,3	0	0,7	8,149233093	12,31624943

View file E evaluation process (Model: Nova Pro 1.0, Temperature = 0)

Run	Regt 1	Regt 2	Regt 3	Total	LOC	Time
View_E_ZSL_run_1	1	0	0	1	41	6
View_E_ZSL_run_2	1	0	0	1	40	5
View_E_ZSL_run_3	1	0	0	1	40	5
View_E_ZSL_run_4	1	0	0	1	42	6
View_E_ZSL_run_5	1	0	0	1	42	6
View_E_ZSL_run_6	1	0	0	1	40	5
View_E_ZSL_run_7	1	0	0	1	40	6
View_E_ZSL_run_8	1	0	0	1	40	5
View_E_ZSL_run_9	1	0	0	1	51	7
View_E_ZSL_run_10	1	0	0	1	46	6
Average	1	0	0	1	42,2	5,7
View_E_VAPU_run_1	1	0	0	1	32	59
View_E_VAPU_run_2	1	0	0	1	35	54
View_E_VAPU_run_3	1	1	1	3	48	50
View_E_VAPU_run_4	1	0	0	1	43	58
View_E_VAPU_run_5	1	0	0	1	45	60
View_E_VAPU_run_6	1	0	0	1	69	62
View_E_VAPU_run_7	0	0	0	0	34	68
View_E_VAPU_run_8	1	0	0	1	66	60
View_E_VAPU_run_9	0	0	0	0	71	72
View_E_VAPU_run_10	0	0	0	0	39	47
average	0,7	0,1	0,1	0,9	48,2	59

View file F evaluation process (Model: Nova Pro 1.0, Temperature = 0)

Run	Reg 1	Reg 2	Reg 3	Total	RF	LOC	Time
View_F_ZSL_run_1	0	0	0	0	3	47	5
View_F_ZSL_run_2	0	0	0	0	3	42	6
View_F_ZSL_run_3	0	0	0	0	3	44	6
View_F_ZSL_run_4	0	0	0	0	4	36	5
View_F_ZSL_run_5	0	0	0	0	4	37	4
View_F_ZSL_run_6	0	0	0	0	0	88	10
View_F_ZSL_run_7	0	0	0	0	2	74	10
View_F_ZSL_run_8	0	0	0	0	4	37	5
View_F_ZSL_run_9	0	0	0	0	2	79	10
View_F_ZSL_run_10	0	0	0	0	4	37	5
Avarage	0	0	0	0	2,9	52,1	6,6
View_F_VAPU_run_1	1	1	0	2	4	41	50
View_F_VAPU_run_2	0	0	0	0	3	48	60
View_F_VAPU_run_3	0	0	0	0	3	35	71
View_F_VAPU_run_4	1	1	0	2	4	45	69
View_F_VAPU_run_5	1	0	0	1	4	33	66
View_F_VAPU_run_6	0	0	0	0	4	38	47
View_F_VAPU_run_7	1	1	0	2	4	40	62
View_F_VAPU_run_8	0	0	0	0	1	63	95
View_F_VAPU_run_9	0	0	0	0	4	36	74
View_F_VAPU_run_10	1	1	1	3	1	88	100
Avarage	0,5	0,4	0,1	1	3,2	46,7	69,4

APPENDIX B: GITHUB UPDATING PROMPTS

This appendix lists the used prompts in each GitHub file updated with both VAPU and with the compared prompt. Text surrounded by '()' means that the prompt was only used in certain LLMs to make it return the whole code.

Table B.1. *Used ZSL prompts by file 1/2*

Reposition/File	Prompt
python-beginner-projects [75] reciept generator/main.py	Add necessary receipt information to receipt generator.
Simple calculator [70] SimpleCalculatorPyQt1.py	Update PyQt5 to PyQt6 in a Python file
python-snake [72] snake.py	Replace python console implementation to PyQt6
The Algorithms - Python [71] search_books_by_isbn.py	Change Python desired_keys dictionary to be constructed by ol_book_data. Replace Python file's console implementation with PyQt6.
The Algorithms - Python [71] password.py	Add a new code generating function to the Python file which always fits to the requirements of is_strong_password functions, return the original code as well.
Python-Mini-Projects [73] Alarm clock/alarm_clock.py	Replace numeric lists to use for loops. Replace tkinter to pyside6. Change drop-down selectors to number pickers in the user interface.
Python-Mini-Projects [73] Bouncing_ball_simulator /ball_bounce.py	Add collision check for ball objects in pygame game. Add possibility to add or reduce balls. Add pause option.
Amazing-Python-Scripts [76] Address-Book /addressbook.py	Change tkinter to PySide6. Fix problems when new address book people to sqlite3 are not properly deleted or found it the database. Add home address and email as optional fields to the address book
python-beginner-projects [75] Organize_Directory /organizer.py/main.py	Replace console implementation with Pyside6. Add an option to select from the organized file types given in the file.
Python-Mini-Project [74] Simple_Http_Server /mhttp.py	Replace server implementation with http.server Python module
Amazing-Python-Scripts [76] Restaurant-Management-System /Restaurant-Management-System.py	Update localtime to be updated every second in the tkinter view. Implement function for button total based on the rest of view.

Table B.2. *Used ZSL prompts by file 2/2*

Reposition/File	Prompt
Python-Mini-Project [74] Finance_Tracker /main.py	Change tkinter and customtkinter to Pyside6. Change matplotlib to plotly. Add possibility to remove income/expenses
Python-GUI-Project [78] Notepad /notepad.py	Change tkinter to Kivy. Add option to convert text to pdf.
Amazing-Python-Scripts [76] Health_Log_Book /main.py & database.py	Add a possibility for calory estimation for both exercise and eating. Add matplotlib chart for the calories.
python-beginner-projects [75] Scientific-Calculator /Scientific_CalC.py	Change tkinter python file to pyside6. Add error handling with all found math errors.
Python-Mini-Project [74] Lazy_Pong /pong.py	Move parser arguments as a menu before start of the game. Allow pong unit to move left and right in it's own side as well within limits.
Python-GUI-Project [78] cafe management system /cafe-management.py	Change tkinter python file into PyQt6. Add ability to change prices in the user interface. Add ability to give a discount.Return full codebase.
Python-Mini-Projects [73] Billing_system/Billing_system.py	(no matter what return the whole given code) Add bill search function to include name and phone number also. Add a possibility to edit any searched bill. (Return the whole original code with the additions.)
The Algorithms - Python [71] geometry /geometry.py	Remove doctests from the file. Create user interface with kivy for all methods, return whole code. Add volume caluclation for sphere, return whole code.
Python-Projects [77] Sudoku-Solver /SudokuGUI.py	Add audio annoncement for every given command in sodoku made with pygame using pyttsx3.

Table B.3. *Used VAPU prompts by file 1/2*

Reposition/File	Prompt
python-beginner-projects [75] reciept generator/main.py	Requirement1: Add necessary receipt information to receipt generator, no other requirements.
Simple calculator [70] SimpleCalculatorPyQt1.py	Requirement1: Update PyQt5 to PyQt6 in a Python file, no other requirements
python-snake[72] snake.py	Requirement1: Replace python console implementation to PyQt6, no other requirements
The Algorithms - Python [71] search_books_by_isbn.py	Requirement1: Change Python desired_keys dictionary to be constructed by ol_book_data. Requirement2: Replace Python file's console implementation with PyQt6.
The Algorithms - Python [71] password.py	Requirement1: Add a new code generating function to the Python file which always fits to the requirements of is_strong_password function, return the original code as well.
Python-Mini-Projects[73] Alarm clock/alarm_clock.py	Requirement1: Replace numeric lists to use for loops. Requirement2: Replace tkinter to pyside6. Requirement3: Change drop-down selectors to number pickers in the user interface.
Python-Mini-Projects [73] Bouncing_ball_simulator /ball_bounce.py	Requirement1: Add collision check for ball objects in pygame game. Requirement2: Add possibility to add or reduce balls. Requirement3: Add pause option.
Amazing-Python-Scripts [76] Address-Book /addressbook.py	Requirement1: Change tkinter to PySide6. Requirement2: Fix problems when new address book people to sqlite3 are not properly deleted or found it the database. Requirement3: Add home address and email as optional fields to the address book.
python-beginner-projects [75] Organize_Directory /organizer.py/main.py	Requirement1: Replace console implementation with Pyside6. Requirement2: Add an option to select from the organized file types given in the file.
Python-Mini-Project [74] Simple_Http_Server /mhttp.py	Requirement1: Replace server implementation with http.server Python module (,no other requirements.)
Amazing-Python-Scripts [76] Restaurant-Management-System /Restaurant-Management-System.py	Requirement1: Update localtime to be updated every second in the tkinter view. (return whole/rest of the given code as answer) Requirement2: Implement function for button total based on the rest of view.(return whole/rest of the given code as answer)

Table B.4. *Used VAPU prompts by file 2/2*

Reposition/File	Prompt
Python-Mini-Project [74] Finance_Tracker /main.py	Requirement1: Change tkinter and customtkinter to Pyside6. Requirement2: Change matplotlib to plotly. Requirement3: Add possibility to remove income/expenses
Python-GUI-Project [78] Notepad /notepad.py	Requirement1: Change tkinter to Kivy. Requirement2: Add option to convert text to pdf.
Amazing-Python-Scripts [76] Health_Log_Book /main.py & database.py	Requirement1: Add a possibility for calory estimation for both exercise and eating, return whole of the given code as answer. Requirement2: Add matplotlib chart for the calories, return whole of the given code as answer.
python-beginner-projects [75] Scientific-Calculator /Scientific_CalC.py	Requirement1: Change tkinter python file to pyside6. Requirement2: Add error handling with all found math errors, return full codebase.
Python-Mini-Project [74] Lazy_Pong /pong.py	Requirement1: Move parser arguments as a menu. before start of the game, return full codebase. Requirement2: Allow pong unit to move left and right in it's own side as well within limits, return full codebase.
Python-GUI-Project [78] cafe management system /cafe-management.py	Requirement1: Change tkinter python file into PyQt6. Requirement2: Add ability to change prices in the user interface, return full codebase. Requirement3: Add ability to give a discount, return full codebase.
Python-Mini-Projects [73] Billing_system/Billing_system.py	Requirement1: Add bill search function to include name and phone number also, return full codebase. Requirement2: Add a possibility to edit any searched bill, return full codebase.
The Algorithms - Python [71] geometry /geometry.py	Requirement1: Remove doctests from the file. Requirement2: Create user interface with kivy for all methods, return whole code. Requirement3: Add volume caluclation for sphere, return whole code.
Python-Projects [77] Sudoku-Solver /SudokuGUI.py	Requirement1: Add audio annoncement for every given command in sodoku made with pygame using pyttsx3. (Add) No other requirements.