

UNIVERSITY OF TARTU  
Faculty of Science and Technology  
Institute of Computer Science  
Computer Science Curriculum

Sander Sats

# Control Code Comprehension with Large Language Models (LLMs)

Master's Thesis (30 ECTS)

Supervisor: Kallol Roy, PhD

Tartu 2024

# Large Language Models for Control System Code Analysis

## Abstract:

Large Language Models (LLMs) are used ubiquitously in many applications ranging from text summarization, automated theorem proving, code comprehension, and generation. Instructions (a.k.a prompt) are used to tune the large language models (LLMs) on solving new downstream tasks and thus enhance the generalization capability. To date, less effort has been given to using LLMs in software engineering, e.g., NL-to-Code tasks, code repair, code comprehension, etc. Comprehending code through LLMs is a non-trivial problem, as structural differences exist between code (program) and natural language. Moreover, commercial LLMs trained on the text corpus are closed-source, often lacking transparency and reproducibility. To fill this gap, this thesis proposes a novel method of tuning instructions (prompt engineering) of the LLMs for solving the code comprehension task. We have optimally tuned GPT-3.5 and GPT-4 to comprehend the Matlab code of control systems engineering. Our handwritten Matlab code simulates the behavior of various control systems, e.g., feedback control and PID control. We propose and design three types of prompts: (i) text prompt, (ii) logical prompt, and (iii) numerical prompt to assist the LLM in comprehending the Matlab code for control systems. We propose a new metric as an evaluation measure to check the correctness of understanding of LLM while solving the task. This thesis research findings show while LLMs (GPT 3.5) are good for solving language tasks, they are not yet mature enough to solve code comprehension of Matlab scripts that work primarily in numerical and linear algebra domains.

## Keywords:

Large Language Models, Prompt Engineering, Natural Language Processing, Control Systems Engineering, Matlab, Feedback Loop, PID controller

**CERCS:** P170 Computer science, numerical analysis, systems, control

## Suurte keelemodelite kasutamine kontrollüsteemi koodi analüüsiks

### Lühikokkuvõte:

Suured keelemudelid (LLM) leiavad kõiksugustes rakendustes laialdaselt kasutust, alates teksti kokkuvõtetest kuni automatiseeritud teoreemide tõestamiseni ja koodi mõistmisest kuni koodi genereerimiseni. Juhiseid (viipasid) kasutatakse, et suuri keelemudeleid (LLM) uute ülesannete lahendamiseks häälestada ja selle läbi üldistusvõimet suurendada. Praeguseini on vähem pingutusi suunatud, et suuri keelemudeleid (LLM) tarkvara arenduses, näiteks loomulikust keelest koodiks, koodi paranduse, koodist arusaamise jne ülesanneks, kasutada. Koodi mõistmine keelemodelite (LLM) abiga ei ole triviaalne probleem, sest programmikoodi ja loomulike keele vahel on struktuurilisi erinevusi. Veel enam, teksti korpuste peal treenitud kaubanduslikud keelemudelid ((LLM) on suletud lähtekoodiga ning neil jääb läbipaistvusest ja korratavusest vajaka. Et seda lünka täita pakub antud töö keelemodelite (LLM) koodi mõistmiseks kasutamise viipade häälestamiseks

välja uudse meetodi. Me olen viiba abil ChatGPT 3.5 ja ChatGPT 4 häälestanud kontroll-süsteemide Matlabi koodi mõistmiseks. Meie käsitsi kirjutatud Matlabi kood simuleerib erinevate kontrollsüsteemide, näiteks tagasiside kontroll ja PID kontroll, käitumist. Me pakume välja ja disainime kolme eri tüüpi viipasid: (i) tekstiline viip, (ii) loogiline viip ja (iii) numbriline viip, et aidata keelemudelil (LLM) kontrollsüsteemide koodist aru saada. Me pakume keelemudelile (LLM) koodi mõistmise õigsuse hindamiseks välja uue mõõdiku. See töö näitab, et kuigi keelemudelid on keeleliste ülesannete lahendamises väga head, siis ei ole nad kontrollkoodi skriptide, mis töötavad peamiselt numbrilistes ja lineaar-algebralistes domeenides, mõistmiseks veel piisavalt küpsed.

**Võtmesõnad:**

Suured keelemudelid, Käsu kavandamine, Loomuliku keele töötlus, Kontrollsüsteemide kavandamine, Matlab, Tagasiside ahel, PID kontroll

**CERCS:** P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Feedback in Control Systems . . . . .	6
1.2	State Space Modelling and Analysis . . . . .	8
1.2.1	The State of a System and State Equation of Continuous Time System . . . . .	8
1.2.2	Vector Matrix Differential Equation of Continuous Time System . . . . .	8
1.2.3	Laplace Transform Solution of State Equations [Pal22] . . . . .	9
1.3	Control System and MATLAB . . . . .	10
1.3.1	Frequency Response . . . . .	10
1.3.2	Liner Time-Invariant System (LTI) . . . . .	11
<b>2</b>	<b>Code comprehension</b>	<b>12</b>
<b>3</b>	<b>Proposed methods</b>	<b>14</b>
3.1	Prompt engineering . . . . .	14
3.2	Experiments . . . . .	15
3.3	New Metric . . . . .	16
3.3.1	Counting reused tokens . . . . .	17
3.3.2	Ground truth . . . . .	18
3.3.3	Cosine similarity . . . . .	18
3.4	Results . . . . .	19
3.4.1	Code example 1 results (Figure 9) . . . . .	19
3.4.2	Code example 2 results (Figure 10) . . . . .	20
3.4.3	Code example 3 results (Figure 11) . . . . .	20
3.4.4	Code example 4 results (Figure 12) . . . . .	21
3.4.5	Code example 5 results (Figure 13) . . . . .	21
3.4.6	Results combined over all code examples . . . . .	22
3.5	Code examples . . . . .	22
3.5.1	Example 1 of plant controller MATLAB code . . . . .	22
3.5.2	Example 2 of plant controller MATLAB code . . . . .	24
3.5.3	Example 3 of plant controller MATLAB code . . . . .	28
3.5.4	Example 4 of plant controller MATLAB code . . . . .	32
3.5.5	Example 5 of plant controller Python code . . . . .	36
<b>4</b>	<b>Inference</b>	<b>39</b>
4.1	Understanding the simplest code . . . . .	39
<b>5</b>	<b>Conclusion and Future Work</b>	<b>43</b>

<b>References</b>	<b>45</b>
<b>Appendix</b>	<b>46</b>
I. Code repository . . . . .	46
II. Licence . . . . .	47

# 1 Introduction

Control Systems have played a crucial role in the advancement of science and engineering. Control systems find ubiquitous applications from space vehicle systems, missile guidance systems to manufacturing and industrial process control. Modern day computers are the integral part of control system engineering development. Control engineering is applicable in various fields of engineering and it's use is so widespread in technology that not finding a working system control circuitry or provider in it is very unlikely. Without advances in control system design a lot of fields like space travel, music systems, industrial production etc would be far from where they are now.

A physical system's behavior can be interpreted as the output of that system in response to some input. Control Systems are used to regulate that response by representing the physical system mathematically in the frequency or time domain. In the context of Control Systems a system is defined as a set of interconnected objects where the objects and attributes have a specific relationship. Any part of a system that we want to control is called a plant or process. The input is applied to the plant and produces the output so to get the desired output we need to send the appropriate input (Figure 1 (a)). A controller is added between the plant and the input when disturbances occur in the system to compensate for the disturbances (Figure 1 (b)). The controller's output signal is the correct input signal for the plant to get the desired output.

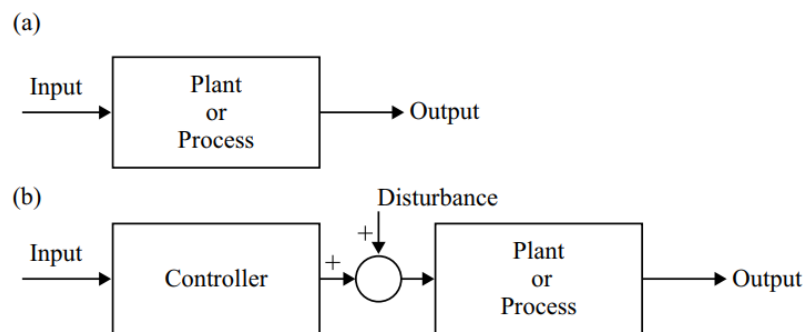


Figure 1. (a) A plant without a controller and (b) A plant with a controller [Pal22].

## 1.1 Feedback in Control Systems

An open loop system is defined as the system in which the control action (actuating signal) is independent of the output or the desired result [Pal22]. Using a simple D.C. motor speed control system (Figure 2) as an example the speed of rotation  $\omega$  (output of the system) depends on the voltage  $V_a$  (input that causes control action). If the load

increases, then the speed  $\omega$  decreases unless the voltage  $V_a$  is also increased. And also in reverse, if the load decreases, then the speed  $\omega$  increases unless the voltage  $V_a$  is also decreased. Having to adjust this manually will require constant attention from the operator.

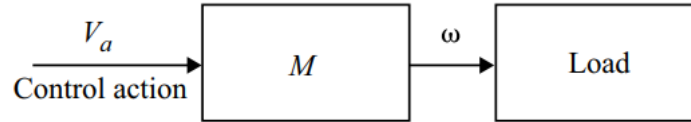


Figure 2. Block diagram representation of a speed control system [Pal22].

However when a D.C. tachogenerator (a device that generates voltage proportional to speed)  $T$  is connected to the motor and it's output  $V_T$  is compared to the input voltage  $V_a$  then the system becomes a feedback or closed loop system(Figure 3). The difference  $V_a - V_T$  (also called Actuating Signal or Error Signal) can be used to increase or decrease the input voltage in order to maintain a constant speed  $\omega$  without the operator having to manually adjust.

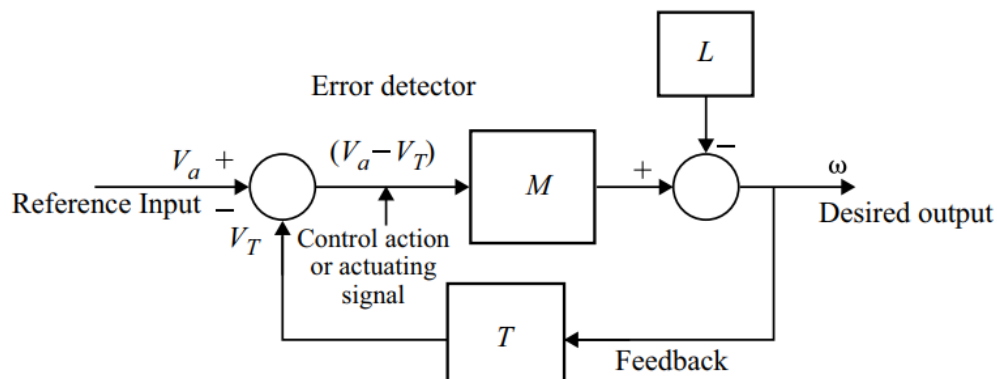


Figure 3. Block diagram of a closed loop D.C. motor speed control system [Pal22].

A system can also be designed with positive feedback where the Actuating Signal becomes  $V_a + V_T$ . In this case, instead of rejecting load side disturbances and maintaining a steady speed, it would mean that the system exaggerates the disturbances. If the load increases, then  $\omega$  decreases. This then decreases  $V_t$  which in turn decreases  $\omega$  which again decreases  $V_t$  until the speed finally becomes zero. If the load is reduced, then  $\omega$  increases which in turn increases  $V_t$  which in turn increases  $\omega$ . This cumulative increase in speed

may reach dangerous levels and damage the system and is called an unstable mode of operation. In practice only negative feedback is employed (except for selective cases like oscillators). A closed loop system is defined as a system in which the measured output is compared with the reference input. The control action depends upon the changes in the output [Pal22].

## 1.2 State Space Modelling and Analysis

The transfer function (T.F.) model was used to analyse linear time invariant continuous time systems, but it is severely limited because it is expressed as a ratio of output to input variables. This means that the internal system behavior is hidden. The T.F. model also requires initial conditions to be zero and cannot handle non-linear, time varying and MIMO (multi-input and multi-output) or large-scale complex systems.

These limitations are remedied by using a state space model where  $n$  first order equations are used in a specific format to represent a differential equation model. This representation completely describes the behavior of all the physical variables in the system and is valid for linear, non-linear and time varying systems even when initial conditions are not zero.

### 1.2.1 The State of a System and State Equation of Continuous Time System

For a linear continuous time system, the state of a system is defined as the minimum number of initial conditions that must be specified at any initial time  $t_0$  so that the complete dynamic behaviour of the system at any time  $t > t_0$  is determined when the input  $u(t)$  is known [Pal22].

This means that the future states of the system are also determined by and change with the input  $u(t)$ . These are called state variables since they track the state in time. The number of state variables varies depending on the physical system:  $n$  state variables are required for a  $n$ th order differential equation and define the  $n$  dimensional state space. If there are less than  $n$  state variables then the system is not fully represented. If there are more, then some of the state variables are redundant and can be derived from others.

### 1.2.2 Vector Matrix Differential Equation of Continuous Time System

State variable equations are expressed in time using compact vector matrix notations. These equations are called vector matrix differential equations [Pal22] and their standard form is shown in (Figure 4).

The state equation(1) and the output equation(2) combined describe the system dynamics. The equations can also be represented as a block diagram (Figure 5).

The dimension of matrices  $A$ ,  $B$  and  $C$  depend on the dimensions of the vectors  $x$ ,  $u$  and  $y$ . In most applications,  $D = 0$  as the direct input  $u(t)$  to output  $y(t)$  transmission is



$$\dot{x}(t) = Ax(t) + Bu(t) \quad (1)$$

$$y(t) = Cx(t) + Du(t) \quad (2)$$

Figure 4. Vector matrix differential equations [Pal22]. Symbol descriptions:  $\dot{x}(t)$  = State vector ( $n \times 1$  dimension);  $y(t)$  = Output vector ( $p \times 1$  dimension);  $u(t)$  = Input vector ( $r \times 1$  dimension);  $A$  = State matrix ( $n \times n$  dimension);  $B$  = Input matrix ( $n \times r$  dimension);  $C$  = Output matrix ( $p \times n$  dimension);  $D$  = Direct transmission matrix ( $p \times r$  dimension)

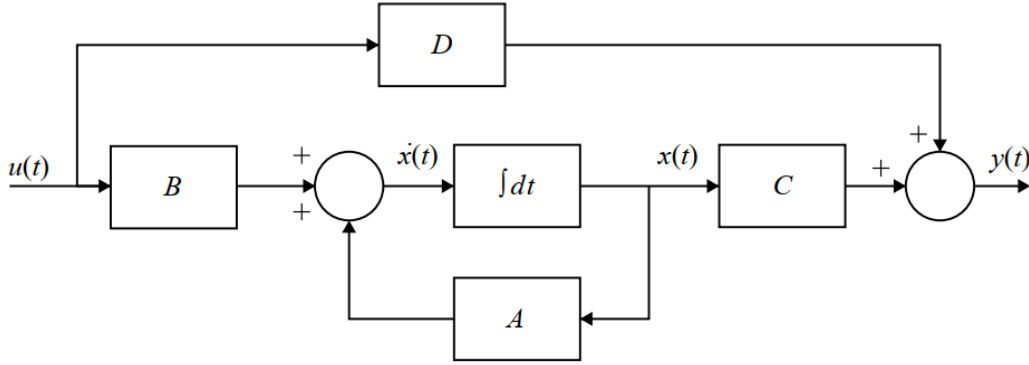


Figure 5. Block diagram of state equations (1) and (2) [Pal22].

not done. It should be observed that only the first derivative of  $x(t)$  appears on the left side of the equation and no derivative of  $x(t)$  appears on the right side.

The state equations are solved in both time and frequency domains. In the frequency domain, a Laplace Transform method is used.

### 1.2.3 Laplace Transform Solution of State Equations [Pal22]

When we take the Laplace transform on both sides of the state equation (1) of the vector matrix differential equations (Figure 4), we get

$$\begin{aligned} sX(s) - x(0) &= AX(s) + BU(s) \\ (sI - A)X(s) &= x(0) + BU(s) \end{aligned} \quad (3)$$

after pre-multiplying both sides of equation (3) by  $[sI - A]^{-1}$ , we get

$$\begin{aligned} X(s) &= [sI - A]^{-1}[x(0) + BU(s)] \\ &= \phi(s)[x(0) + BU(s)] \end{aligned} \quad (4)$$

where

$$\begin{aligned}\phi(s) &= [sI - A]^{-1} \\ X(s) &= \phi(s)x(0) + \phi(s)BU(s)\end{aligned}\tag{5}$$

then we take the inverse Laplace transform and get

$$x(t) = L^{-1}[\phi(s)x(0)] + L^{-1}[\phi(s)BU(s)]\tag{6}$$

$\phi(s)$  defined in equation (5) is the state-transition matrix (STM). In equation (6),  $L^{-1}[\phi(s)x(0)]$  gives the zero input response and  $L^{-1}[\phi(s)BU(s)]$  gives the zero state response.

### 1.3 Control System and MATLAB

The main objective in control systems is to regulate the behavior of a physical system in terms of the input to that system. The behavior of a physical system can be viewed as the output of that system and can be represented mathematically in the time or frequency domain. A physical system needs to be converted to a mathematical representation to implement the concepts of control systems. The opposite is also sometimes necessary.

MATLAB (Matrix Laboratory) is an environment for engineers of all domains to create algorithms and models to use programming capabilities to compute and analyse numerical data. It was originally intended to perform operations on vectors and matrices, but several toolboxes that enable operating on control systems, signal/image processing, deep learning, robotics and so on have been developed over the years. MATLAB is a high-level programming language and the platform offers high graphic capability for visualizing in 2D and 3D.

#### 1.3.1 Frequency Response

Control Systems behavior (system response) is characterized by a quantity called transfer function. The transfer function is computed either in the time or frequency domain. The transfer function  $H(s)$  in the frequency domain ( $s = \sigma + i\omega$ ) is given as the ratio of two polynomials that shows the behavior of the system at different frequency components given by [Pal22]:

$$\begin{aligned}H(s) &= \frac{Y(s)}{X(s)} \\ &= \frac{L(x(t))}{L(y(t))}\end{aligned}\tag{7}$$

where the real part of the  $Re(s) = \sigma$  gives the attenuation and the imaginary part  $Im(s) = \omega$  shows the oscillations,  $L$  is the Laplace transform,  $x(t)$  is the input function and  $y(t)$  is output function.

### 1.3.2 Liner Time-Invariant System (LTI)

Linear time-invariant systems have two distinctive characteristics: linearity and time invariance [Hos22]. Linearity here refers to the final output of the control system is the sum of the effect of individual input applied to it (Figure 6). Time invariance means that the output is not dependent on the time the input is applied - if the same input is applied at different timings, the output will still be the same.

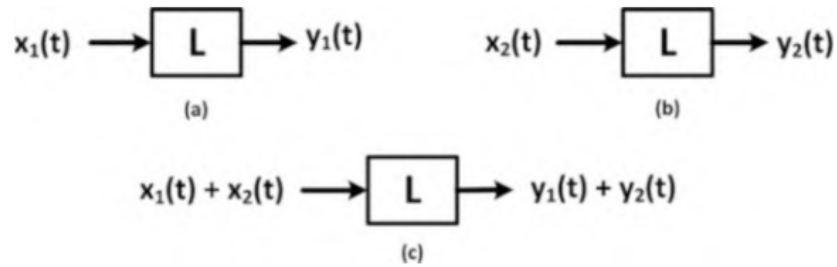


Figure 6. Illustration of linearity. Outputs change linearly according to inputs, summing the inputs  $x_1(t)$  and  $x_2(t)$  results in summing the outputs  $y_1(t)$  and  $y_2(t)$  [Hos22].

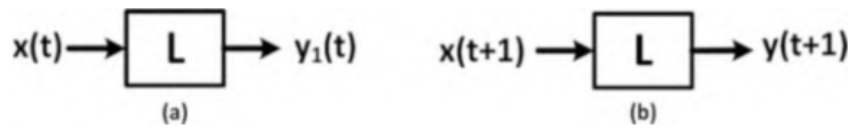


Figure 7. Illustration of time invariance. If a system produces output  $y(t)$  for input  $x(t)$  and the input is shifted to  $x(t + 1)$  then the output is also shifted to  $y(t + 1)$  [Hos22].

## 2 Code comprehension

Using Large Language Models (LLMs) for code generation is not a new idea and has recently gained considerable attention but existing models are still not very good at it. The most promising approaches are based on transformer-based large-scale language models which have been very successful in the text domain [YLZ<sup>+</sup>23, CHW24].

These models, such as Codex [CTJ<sup>+</sup>21] and AlphaCode [LCC<sup>+</sup>22], generate code by applying a transformation from natural language descriptions into programming language and this has worked for Python, Java and C languages. Though the passing rates are fairly low (Figure 8): Codex gets 24.52% / 3.08% [CTJ<sup>+</sup>21] and AlphaCode gets 20.36% / 7.75% [LCC<sup>+</sup>22] on introductory-level/competition-level tasks from APPS dataset [HBK<sup>+</sup>21]. One of the main culprits is their lack of awareness of program semantics. Because of the way the transformers treat the description and code as token sequences the deep semantic features of programs are lost. This can manifest itself making the model fixate on some keywords while ignoring the rest of the context.

Model	Introductory-level tasks	Competition-level tasks
Codex	24.52%	3.08%
AlphaCode	20.36%	7.75%

Figure 8. Performance of Codex [CTJ<sup>+</sup>21] and AlphaCode [LCC<sup>+</sup>22] in code generation tasks on the APPS dataset [HBK<sup>+</sup>21].

This issue applies double fold as this loss of deep semantic features happens when trying to understand the natural language description as well. Generating a program requires understanding the entire task description which usually contains complex logic and dependencies.

A recent study has discovered that automatically generated code shares common programming mistakes with human-crafted solutions (which makes sense as these models are trained on human-crafted code) and that by applying automated program repair (APR) to model generated code, the results can be improved [FGM<sup>+</sup>23]. They identified that the main issues were incomplete code, invoking undefined variables/functions/classes and using the wrong algorithms. They also found that auto-generated programs contain certain anti-patterns: variable names indicate wrong algorithms, repeated similar code blocks and irrelevant helper functions. The main reasons for these issues are limited search space, inability to produce multi-edit patches and lack of awareness of program dependencies.

For code comprehension we try to turn the problem around and instead of the input being the natural language description and the output being the code, we instead use the code as the input and expect natural language description as the output.

In the case of MATLAB code in the Control Code domain, this issue of losing deep semantic features in token sequence transformation could be even worse, because this language is very succinct - it packs a lot of semantics into very little syntax. Or the issue could be mitigated instead. The more succinct syntax could help the transformer understand the meaning better, because it's not so distributed, so the training examples "pack a bigger punch". Also the context of Control Code may give the models a narrower path to traverse (so they don't get sidetracked by confusing tokens and remain in context) as the control approaches are quite standardized and follow a similar flow.

### 3 Proposed methods

In this work we propose a method to use ChatGPT to explain, analyse and understand MATLAB code. We use examples of code with varying levels of complexity and prompts with differing levels of additional relevant information to check how this affects the performance. We have designed a prompt and error metric which allow us to use ChatGPT 4's image input feature to measure code understanding. This is done by using code examples that create MATLAB plots and have ChatGPT first describe the resulting plot from the code and compare that result with description of the plot from an image.

#### 3.1 Prompt engineering

The prompt has been engineered based on the Persona Pattern [WFH<sup>+</sup>23] to make ChatGPT act as an expert MATLAB coder and analyst. Initially the idea was to use a minimal prompt as a baseline, but it was hopelessly bad. The prompt has been designed to give the best response and to be able to understand where it went wrong by using different reasoning steps to push it into a space where the next tokens will form a coherent and correct response.

The prompt also has some additional guiding lines that were added during trial and error and a dynamic section for the explanation which is varied for different experiments. The base prompt is shown in (Listing 1).

```
Act as an expert in control theory that has decades of
  experience writing control code scripts in MATLAB.
These expert skills include but are not limited to
  recognizing and intuiting the shape of the line a piece of
  code will produce.
Pay special attention to how the variables can change over
  iterations.
The user query will be some MATLAB code. Your task will be to
  analyse the code following the reasoning steps and to
  describe the resulting plot.
When asked for values, don't explain where these values can
  be obtained from, just simulate the code and state the
  values as best you can.
{explanation}

The <response to customer> in the final "Response to
  user:{delimiter}" step should strictly follow the provided
  format,
where the text enclosed in <> should be replaced with the
  described value and contain nothing else:
[plot shape: <shape of the plot>, maximum: <maximum value>,
```

```

    minu: <minimum value>, pivot points: <pivot points>,
    direction: <is the graph ascending, descending or mixed>,
    x-intercepts: <points where the plot intercepts the x-axis>
    y-intercepts: <points where the plot intercepts the y-axis>]
.

Step 1:{delimiter} Explain the code line by line.

Step 2:{delimiter} What are the initial and final values on
    for the plot axes?

Step 3:{delimiter} What type of control system is this
    (Open-loop, closed-loop, continuous, discrete, linear,
    nonlinear, time-invariant, time-varying, SISO or MIMO)?

Step 4:{delimiter} Emulate the code step by step and
    calculate all the value changes

Step 5:{delimiter} Describe in words what the plot looks like
    (poles, growth regions, dips etc)

Use the following format:
Step 1:{delimiter} <step 1 reasoning>
Step 2:{delimiter} <step 2 reasoning>
Step 3:{delimiter} <step 3 reasoning>
Step 4:{delimiter} <step 4 reasoning>
Step 5:{delimiter} <step 5 reasoning>
Response to user:{delimiter} <response to customer>

Make sure to include {delimiter} to separate every step and
    don't forget the final response in the correct format
    after the reasoning steps.

```

Listing 1. Base prompt.

## 3.2 Experiments

The experiment design is quite simple: ChatGPT is given instructions to play the role of a developer and describe the plot resulting from the provided MATLAB control code as the system message and the code itself as the prompt. There are 3 different levels of code breakdown the check if this helps ChatGPT perform better:

- textual: just the instruction text with no additional explanation (Listing 1);

- logical: adding a breakdown of how the state-space model function works and what it does internally (Listing 2);
- numerical: same as logical, but additionally the internal variables of the state-space model are calculated and provided (Listing 3).

```
The ss() function is a linear system solver. Calling
  ss(A,B,C,D,ts) creates the discrete-time state-space model
  object of the following form with the sample time ts (in
  seconds):
x[n+1]=Ax[n]+Bu[n]
y[n]=Cx[n]+Du[n]

Where x, u and y represent the states, inputs and outputs
respectively, while A, B, C and D are the state-space
matrices
```

Listing 2. State-space model explanation.

```
For this code, the values after the ss() calls will be:
sysd => A = [[1, 0.1], [0, 1]], B = [0.005, 0.1], C = [1, 0],
        D = 0, ts = 0.1
sysdCL => A = [[0.5, 0.025], [-10, -0.5]], B = [0.5, 10], C =
          [1, 0], D = 0, ts = 0.1
```

Listing 3. State-space model values.

### 3.3 New Metric

The metric equation (8) for measuring the performance of ChatGPT consists of two equally weighted parts. The first part is the ratio of tokens reused from the prompt (more reused words means lower value) - this is to penalize the model for just reusing the prompt itself. The second part calculates the cosine similarity between the ground truth and the response (in this case, the difference between how the model describes the plot from a picture vs how it describes the plot from the code). These values are added together and divided by two to get a value between 0 and 1. The code for calculating the metric is shown in (Listing 4). Only the part of the response in "<response to customer>"



is considered for the error calculation, the reasoning steps are there only for instruction and debugging purposes. The response received from ChatGPT 4 about the plot doesn't have the reasoning steps (and they wouldn't make sense to compare as the reasoning is completely different from code or a graph) so including them would make the comparison useless.

$$E = 0.5 \times [\text{reusedtokens}] + 0.5 \times [\text{cosinesimilarity}] \quad (8)$$

```
1 def get_metric(prompt, response, truth):
2     return 0.5 * count_reused_tokens(prompt, response) + 0.5
   * get_cosine_similarity_by_words(response, truth)
```

Listing 4. Code for calculating the metric in equation (8).

### 3.3.1 Counting reused tokens

Counting the reused tokens is done fairly simply (Listing 5): first we use the tiktoken package to tokenize the texts, then we remove duplicate words (convert to sets) and count the words that were reused. Finally we calculate the ratio and invert it so it is compatible with cosine similarity (when comparing against ground truth, more similarity means higher value, but when comparing against the prompt, more similarity means lower value).

```
1 import tiktoken
2
3 def tokenize(text, model="gpt-3.5-turbo"):
4     return
       tiktoken.encoding_for_model("gpt-3.5-turbo").encode(text)
5
6 def count_reused_tokens(a, b):
7     a_tokenized = set(tokenize(a))
8     b_tokenized_set = set(tokenize(b))
9
10    count = 0
11    for token in a_tokenized:
12        if token in b_tokenized_set:
13            count += 1
14
15    return 1 - count / len(a_tokenized)
```

---

Listing 5. Code for counting reused tokens.

### 3.3.2 Ground truth

The ground truth for the MATLAB control codes is obtained by using the image input capability of ChatGPT 4. The control code snippets are run in MATLAB and ChatGPT 4 Grimoire [CSY<sup>+</sup>24], which is a specially trained expert coder instance of ChatGPT 4, is asked to describe a screenshot of the resulting plot. We tried to design the prompt (Listing 6) in such a way that the response from the image and prompt would be as similar as possible to the one from the code and prompt.

```
Please describe this graph as accurately as you can. Your
response should strictly follow the provided format, where
the text enclosed in <> should be replaced with the
described value and contain nothing else:
[plot shape: <shape of the plot>, maximum: <maximum value>,
minimum: <minimum value>, pivot points: <pivot points>,
direction: <is the graph ascending, descending or mixed>,
x-intercepts: <points where the plot intercepts the x-axis>
y-intercepts: <points where the plot intercepts the y-axis>]
```

Listing 6. Prompt for ChatGPT 4 to get the ground truth for plot description.

### 3.3.3 Cosine similarity

In data analysis, cosine similarity is a measure of similarity between two non-zero vectors defined in an inner product space. Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths. It follows that the cosine similarity does not depend on the magnitudes of the vectors, but only on their angle. The cosine similarity always belongs to the interval  $[-1, 1]$ . For example, two proportional vectors have a cosine similarity of 1, two orthogonal vectors have a similarity of 0, and two opposite vectors have a similarity of -1. To calculate the similarity of the embedding vectors, we use OpenAI's API to get the word embeddings for the ground truth and response (Listing 7).

```

1 import numpy as np
2 from numpy.linalg import norm
3
4 def get_cosine_similarity(A, B):
5     return np.dot(A, B)/(norm(A)*norm(B))
6
7 def get_cosine_similarity_by_words(a, b):
8     A = get_openai_word_embeddings(a)
9     B = get_openai_word_embeddings(b)
10
11     return get_cosine_similarity(A, B)

```

Listing 7. Code for calculating cosine similarity.

## 3.4 Results

For each prompt and code example combination, the prompts are run 10 times and the results are averaged.

### 3.4.1 Code example 1 results (Figure 9)

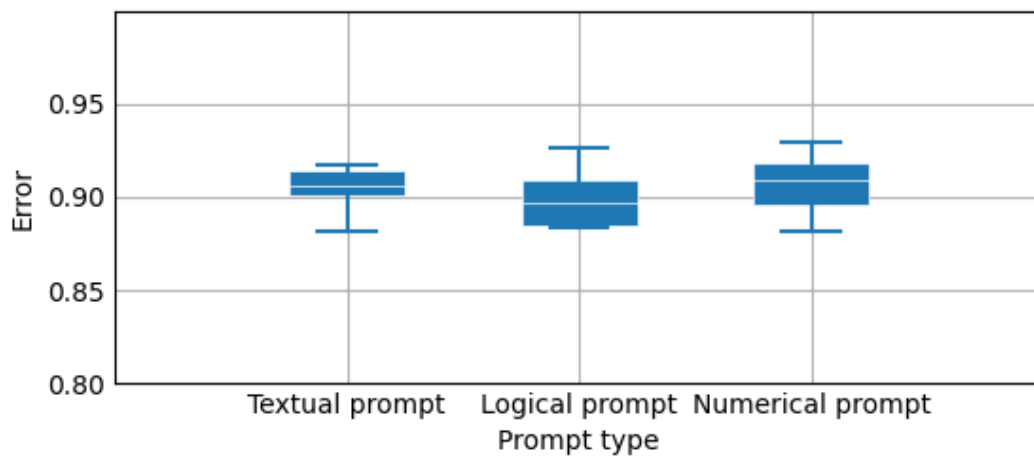


Figure 9. Results for deadbeat controller code example (Listing 8).

### 3.4.2 Code example 2 results (Figure 10)

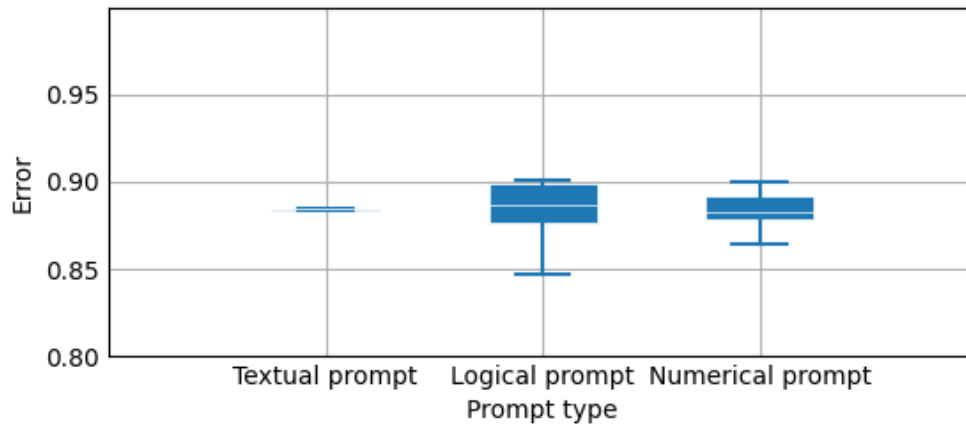


Figure 10. Results for code example 2 (Listing 11).

### 3.4.3 Code example 3 results (Figure 11)

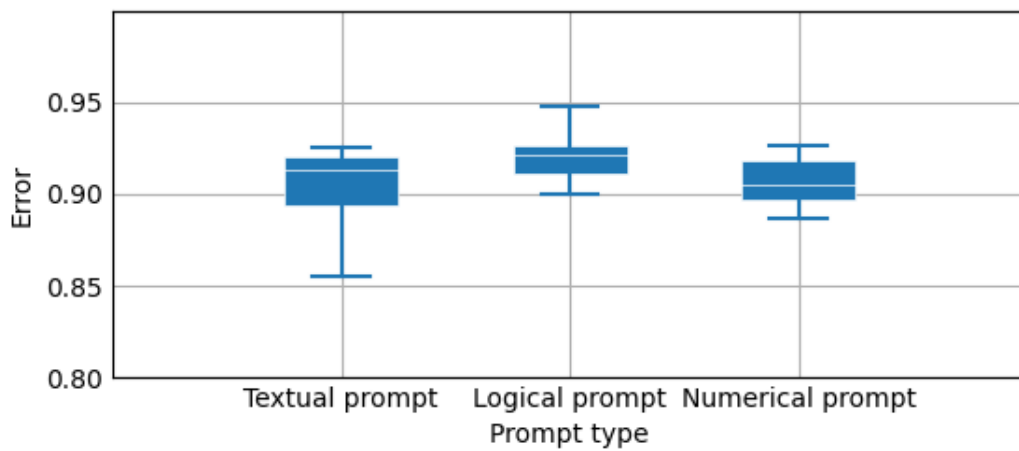


Figure 11. Results for code example 3 (Listing 14).

#### 3.4.4 Code example 4 results (Figure 12)

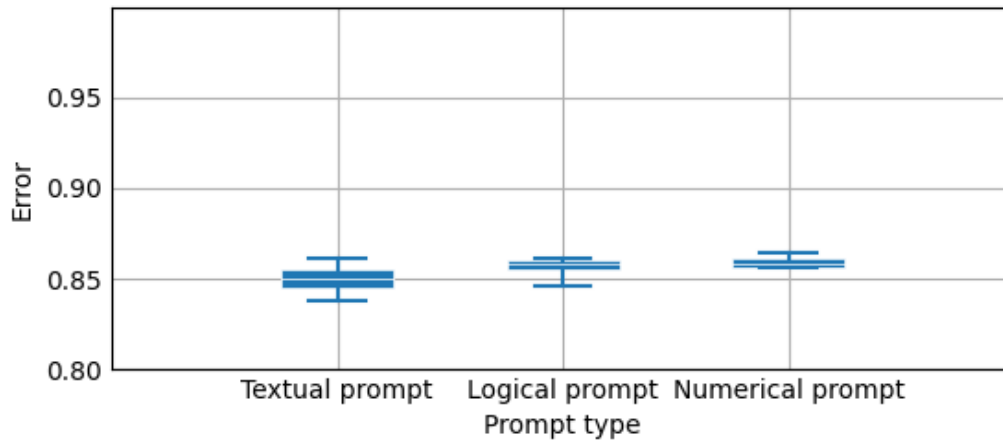


Figure 12. Results for code example 4 (Listing 17).

#### 3.4.5 Code example 5 results (Figure 13)

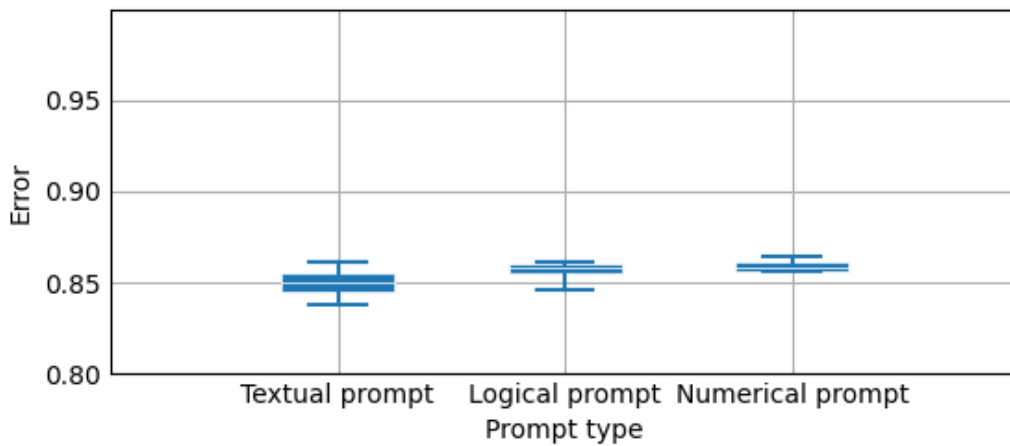


Figure 13. Results for code example 5 (Listing 20).

### 3.4.6 Results combined over all code examples

Finally we combine all the code examples together to see if the prompt improvements had an overall effect on performance and find that it is negligible (Figure 14).

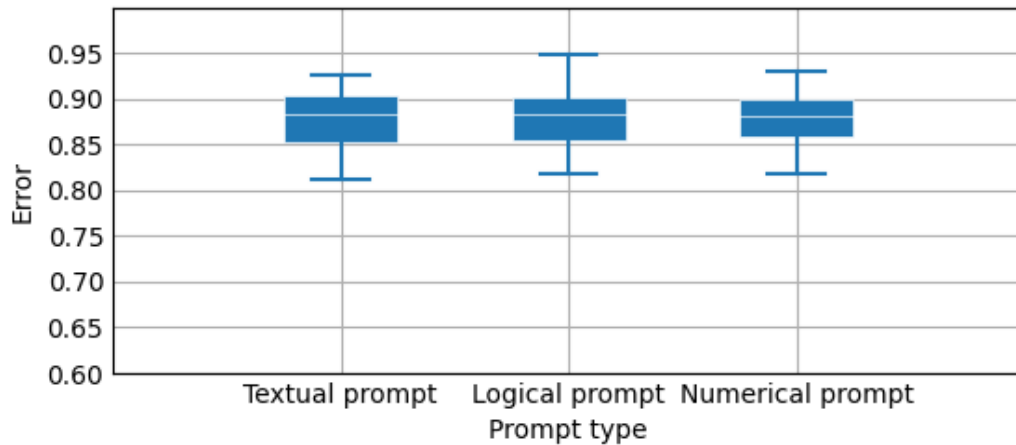


Figure 14. Results combined over all code examples.

## 3.5 Code examples

Here we list all the control code examples with explanations that were used to assess ChatGPT's understanding. The comments in the code are added for reader clarity and were removed when presented to ChatGPT.

### 3.5.1 Example 1 of plant controller MATLAB code

Code example (Listing 8 for a deadbeat controller [NMBM98]).

```
1 // Deadbeat controller
2 // Plant Model
3 T=0.1;
4 F=[1,T;0,1];g=[T^2/2;T];c=[1,0];
5 sysd=ss(F,g,c,0,T);
6 // Pole Placement
7 polesCL=[0,0];
8 k=acker(F,g,polesCL)
9 // Step Response
10 sysdCL=ss(F-g*k,100*g,c,0,'Ts',T);
11 step(sysdCL)
```

Listing 8. Example 1 of plant controller MATLAB code (plot in Figure 15)

And ChatGPT 3.5's response (Listing 9). Showing this example specifically to indicate that sometimes the prompt doesn't work almost at all (notice that the "Response to user" is mostly just spitting out the values without replacement.

Step 1:####

- `T=0.1`: Assigns the value 0.1 to the variable T.
- `F=[1,T;0,1]`: Creates a matrix F with values 1, T, 0, and 1.
- `g=[T^2/2;T]`: Creates a column vector g with elements  $T^2/2$  and T.
- `c=[1,0]`: Creates a row vector c with elements 1 and 0.
- `sysd=ss(F,g,c,0,T)`: Creates a discrete-time state-space system in MATLAB with the state matrix F, input matrix g, output matrix c, and zero-order-hold on the input with a sample time of T.
- `polesCL=[0,0]`: Initializes the desired closed-loop poles.
- `k=acker(F,g,polesCL)`: Calculates the state feedback gain matrix k using the Ackermann method.
- `sysdCL=ss(F-g*k,100*g,c,0,'Ts',T)`: Creates a closed-loop discrete-time state-space system with the state matrix  $F-g*k$ , input matrix  $100*g$ , output matrix c, and zero-order-hold on the input with a sample time of T.
- `step(sysdCL)`: Plots the step response of the closed-loop system.
- `stepinfo(sysdCL)`: Calculates and displays the step response characteristics of the closed-loop system.

Step 2:####

Since the code generates a step response plot, the initial value for the plot axes will be at 0, and the final values will depend on the response characteristics of the system.

Step 3:####

This is a closed-loop, discrete, linear, time-invariant, Single-Input Single-Output (SISO) control system.

Step 4:####

- The state matrix F is  $\begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}$
- The input matrix g is  $\begin{bmatrix} 0.005 & 0.1 \end{bmatrix}$
- The output matrix c is  $\begin{bmatrix} 1 & 0 \end{bmatrix}$
- The state feedback gain matrix k is calculated as  $k = [20,$

```

1]
- The closed-loop state matrix Fcl is [[0.8, -0.005]; [-20,
0]]
- The closed-loop input matrix gcl is [0.5, 10]
- The step response plot is generated with the closed-loop
system.

Step 5:####
The plot shape will be a step response. The system may
exhibit overshoot or settling time based on the poles'
location. The graph will rise initially, potentially
overshoot, and then settle to a final value.

Response to user:####
[plot shape: step response, maximum: <maximum value>,
minimum: <minimum value>, pivot points: <points of
interest>,
direction: mixed, x-intercepts: <points where the plot
intercepts the x-axis>, y-intercepts: <points where the
plot intercepts the y-axis>]

```

Listing 9. Example ChatGPT 3.5 response to deadbeat controller MATLAB code (Listing 8, Figure 15)

And here is Grimoire's [CSY<sup>+</sup>24] response (Listing 10) to the actual plot of the code in MATLAB (Figure 15).

```

[plot shape: step, maximum: 1, minimum: 0, pivot points:
(0.2, 1), (0.4, 0.5), (0.6, 0.5), direction: mixed,
x-intercepts: none, y-intercepts: (0,1)]

```

Listing 10. ChatGPT 4 response to the plot of deadbeat code example (Figure 15).

### 3.5.2 Example 2 of plant controller MATLAB code

MATLAB code (Listing 11) for simulating a closed loop control system for 10 seconds with 0.1 second intervals.

```

1 // Plant parameters (m,b) and time parameter t are set, then
the control input variable (u) is initialised

```



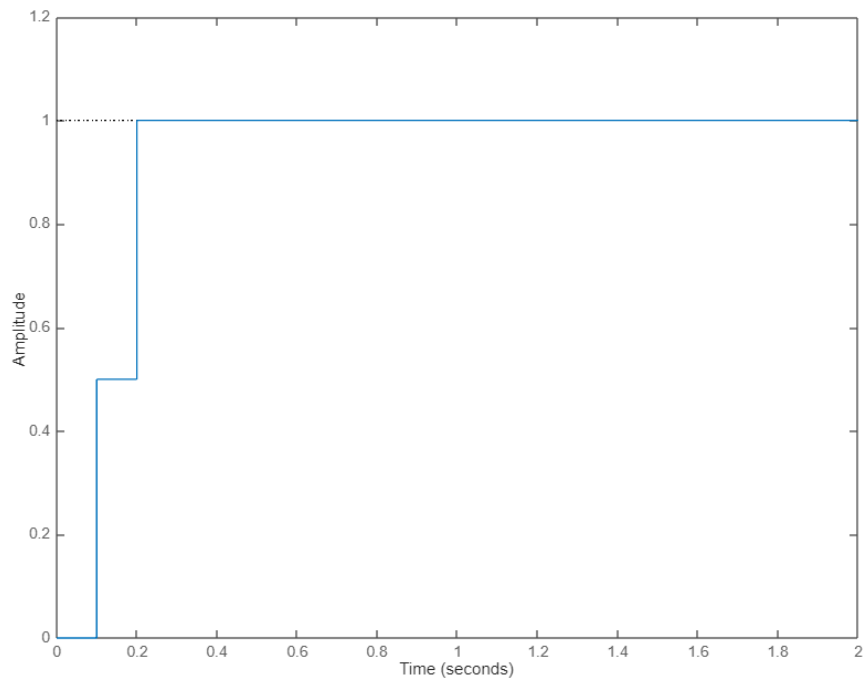


Figure 15. Plot for deadbeat controller code example (Listing 8).

```

2 m = 1000;
3 b = 50;
4 t = 0:0.1:10;
5 u = 500*ones(size(t));
6
7 // Plant dynamic matrices are defined
8 A = [-b/m];
9 B = [1/m];
10 C = [1];
11 D = [0];
12
13 // Get the state space model of the plant by running the ss()
    function
14 sys = ss(A,B,C,D);
15
16 // Initialize the plant system variable
17 x0 = [0];
18
19 // Set the pole position
20 p1 = -1.5;
21
22 // Generate the control gain K using the pole placement
    technique

```

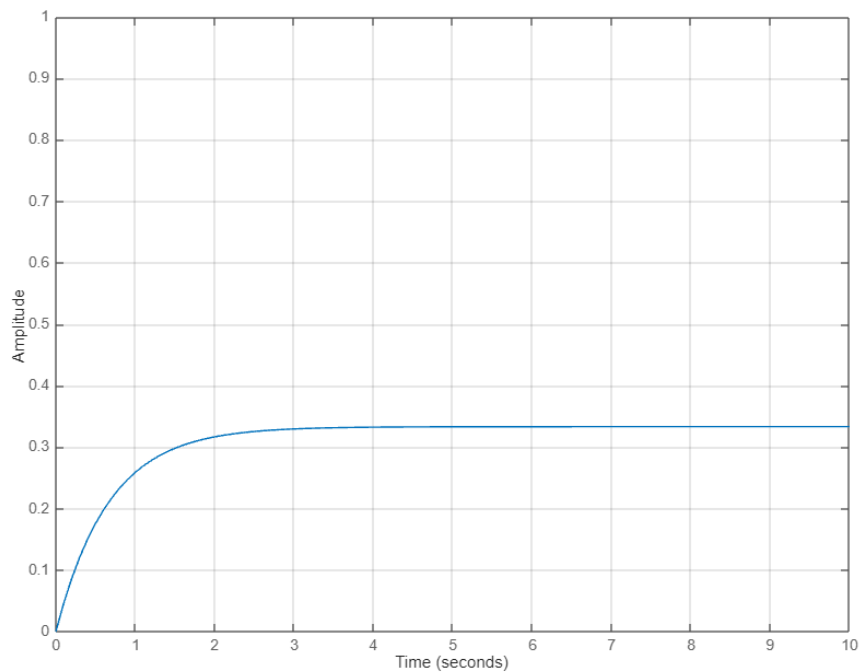


Figure 16. Plot for code example 2 (Listing 11).

```

23 K = place(A,B,[p1])
24
25 // Get the state space model of the closed-loop system (i.e.,
    plant + controller)
26 sys_cl = ss(A-B*K,B,C,D);
27
28 // Simulate the closed-loop systems for 10 seconds starting
    from the initial state valuation
29 lsim(sys_cl,u,t,x0);

```

Listing 11. Example 2 of plant controller MATLAB code (plot in Figure 16)

And ChatGPT 3.5's response (Listing 12). Showing an example of the entire response with the reasoning steps, even though only the final piece is considered for the metric calculation.

```

Step 1:####
- `m = 1000;`: Assigns a value of 1000 to the variable `m`,
    representing the mass.

```

```

- `b = 50;`: Assigns a value of 50 to the variable `b`,
  representing the damping coefficient.
- `t = 0:0.1:10;`: Creates a time vector from 0 to 10 with a
  step size of 0.1.
- `u = 500*ones(size(t));`: Creates an input vector `u` with
  the same size as `t`, filled with 500.

- `A = [-b/m];`: Defines the state matrix `A` for the
  state-space system.
- `B = [1/m];`: Defines the input matrix `B` for the
  state-space system.
- `C = [1];`: Defines the output matrix `C` for the
  state-space system.
- `D = [0];`: Defines the feedthrough matrix `D` for the
  state-space system.

- `sys = ss(A,B,C,D);`: Creates a state-space system `sys`
  with the defined matrices.
- `x0 = [0];`: Sets the initial state vector `x0` to 0.
- `p1 = -1.5;`: Sets the desired pole location for the
  controller.
- `K = place(A,B,[p1]);`: Calculates the state feedback gain
  matrix `K` using pole placement.
- `sys_cl = ss(A-B*K,B,C,D);`: Creates a closed-loop
  state-space system `sys_cl` by incorporating the state
  feedback.

- `lsim(sys_cl,u,t,x0);`: Simulates the response of the
  closed-loop system to the input `u` over time `t` with
  initial state `x0`.

```

Step 2:####

The initial value for the plot axes is 0, and the final value for the plot axes is 10 for the time vector `t`.

Step 3:####

This is a closed-loop control system with state feedback.

Step 4:####

```

- `A = [-b/m]`:  $A = [-50/1000] = [-0.05]$ 
- `B = [1/m]`:  $B = [1/1000] = [0.001]$ 
- `C = [1]`
- `D = [0]`
- `K = place(A,B,[p1])`:  $K = [-0.05 - 0.001*p1] = [-0.05 + 0.001*1.5] = [-0.0485]$ 

```

Step 5:####

The plot shape is a decaying exponential curve starting from the initial state of 0 due to the negative pole placement.

The maximum value is 500, and the minimum value approaches 0. The graph is descending and intersects the x-axis at approximately 6.9 and the y-axis at 0.

Response to user:####  
[plot shape: decaying exponential curve, maximum: 500,  
minimum: 0, pivot points: N/A, direction: descending,  
x-intercepts: 6.9, y-intercepts: 0]

Listing 12. Example ChatGPT 3.5 response to code example 2 of plant controller MATLAB code (Listing 11, Figure 16)

And here is Grimoire's [CSY<sup>+</sup>24] response (Listing 13) to the actual plot of the code in MATLAB (Figure 16).

[plot shape: exponential saturation, maximum: 0.4, minimum:  
0, pivot points: (0, 0) to (3, 0.4), direction: ascending  
then constant, x-intercepts: none, y-intercepts: (0, 0)]

Listing 13. ChatGPT 4 response to the plot of code example 2 (Figure 16).

### 3.5.3 Example 3 of plant controller MATLAB code

MATLAB code for a more complex control system with an integrator (Listing 14).

```

1 // Plant parameters and time parameter t are defined.
2
3 m1 = 2500;
4 m2 = 320;
5 k1 = 80000;
6 k2 = 500000;
7 b1 = 350;
8 b2 = 15020;
9
10 A = [0          1      0
11       -(b1*b2)/(m1*m2)  0
12       ((b1/m1)*((b1/m1)+(b1/m2)+(b2/m2)))-(k1/m1)  -(b1/m1)
13       b2/m2          0  -((b1/m1)+(b1/m2)+(b2/m2))
14                               1

```

```

13      k2/m2          0  -((k1/m1)+(k1/m2)+(k2/m2))
14      B = [0          0
15            1/m1      (b1*b2)/(m1*m2)
16            0          -(b2/m2)
17            (1/m1)+(1/m2)  -(k2/m2) ];
18      C = [0  0  1  0];
19      D = [0  0];
20
21      sys = ss(A,B,C,D);
22      step(sys*[0;1]*.1,0:0.0001:.005);
23
24      // Continuous to discrete conversion of the plant model
25
26      T = .0005;
27      d_sys = c2d(sys,T,'zoh')
28
29      // Adding an integrator to the system in order to drive the
30      // steady-state response to zero
31
32      Ai = 1;
33      Bi = T;
34      Ci = 1;
35      Di = T/2;
36      [As,Bs,Cs,Ds]=ssdata(d_sys);
37      Aa = [As, zeros(4,1);
38            Bi*Cs, Ai];
39      Ba = [Bs;
40            0,0];
41      Ca = [Cs, 0];
42      Da = Ds;
43
44      d_sys_int = ss(Aa,Ba,Ca,Da,T);
45      [Ad,Bd,Cd,Dd] = ssdata(d_sys_int);
46
47      // Designing the controller. We use the place command to
48      // compute the gain matrix, K, which will, in feedback, give
49      // us the desired closed-loop poles.
50
51      sys1 = d_sys_int*[1;0]; //select the first input
52      [num,den] = tfdata(sys1,'v');
53      z = roots(num)
54      p1 = z(1);
55      p2 = z(2);
56      p3 = z(3);
57      p4 = .9992;
58      p5 = .5;
59      K=place(Ad,Bd*[1;0],[p1 p2 p3 p4 p5])

```

```

58 // Simulating the closed-loop response
59
60 d_sys_cl = ss(Ad-Bd*[1;0]*K,Bd,Cd,Dd,T);
61 step(-.1*d_sys_cl*[0;1],5);
62 d_sys_cl = ss(Ad-Bd*[1;0]*K,Bd,Cd,Dd,T);
63 step(-.1*d_sys_cl*[0;1],5);
64 axis([0 0.1 -0.01 0.01])

```

Listing 14. Example 3 of plant controller MATLAB code (plot in Figure 17).

And ChatGPT 3.5's full response (Listing 15).

Step 1:####

The code defines system parameters such as masses ( $m_1$ ,  $m_2$ ), spring constants ( $k_1$ ,  $k_2$ ), damping coefficients ( $b_1$ ,  $b_2$ ), and creates state-space matrices  $A$ ,  $B$ ,  $C$ , and  $D$ . It then creates a state-space system ( $sys$ ) and plots the step response of the system for an input of 0.1 at time instances from 0 to 0.005. It discretizes the system using zero-order hold ( $zoh$ ) method with a sampling time of 0.0005. It then creates an integral system and calculates the poles. Next, it calculates the gain matrix  $K$  using pole placement method and creates a closed-loop system. Finally, it plots the step response of the closed-loop system for an input of -0.1 at time instances from 0 to 5 and sets the axis limits for the plot.

Step 2:####

The initial values for the plot axes are x-axis: 0, y-axis: -0.01 and the final values are x-axis: 0.1, y-axis: 0.01.

Step 3:####

This is a closed-loop discrete-time Single-Input Single-Output (SISO) control system.

Step 4:####

- The poles of the discretized system are calculated as  $z = [0.9992, 0.5, 0.5]$ .
- The gain matrix  $K$  is calculated using the pole placement method.
- The closed-loop system is created using the calculated gain matrix.
- The step response of the closed-loop system for an input of -0.1 is plotted over 5 seconds.

Step 5:####

The plot shape is a step response with multiple steps due to the closed-loop control. The plot starts at  $-0.01$ , rises to a maximum value, then settles at  $0.01$ . The plot has a mixed direction, initially descending and then ascending. The pivot points are where the plot changes direction. The x-intercepts are the points where the plot intercepts the x-axis, and the y-intercepts are the points where the plot intercepts the y-axis.

Response to user:####

[plot shape: step response, maximum:  $0.01$ , minimum:  $-0.01$ , pivot points: multiple, direction: mixed, x-intercepts: multiple, y-intercepts:  $0$ ]

Listing 15. Example ChatGPT 3.5 response to code example 3 of plant controller MATLAB code (Listing 14, Figure 17)

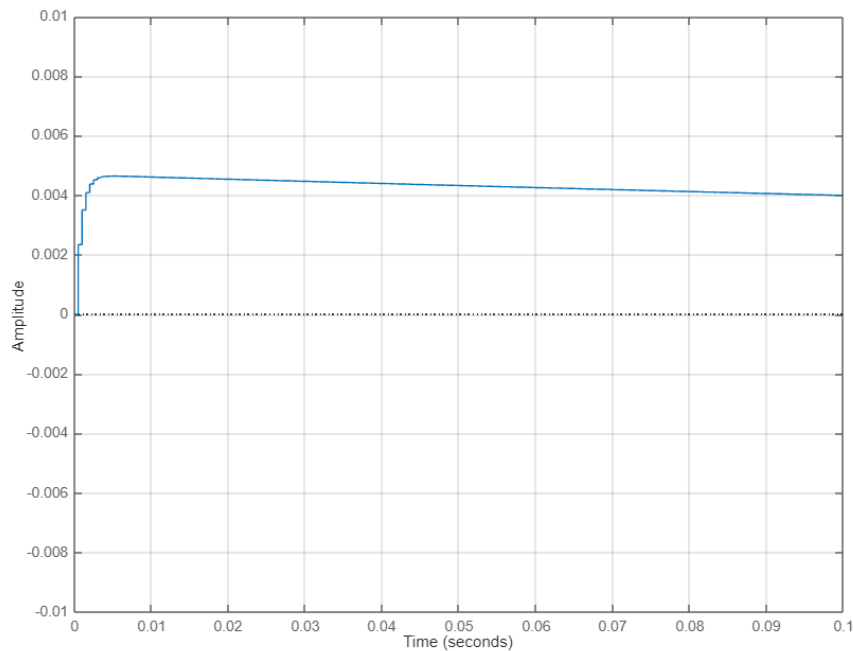


Figure 17. Plot for code example 3 (Listing 14).

And here is Grimoire's [CSY<sup>+</sup>24] response (Listing 16) to the actual plot of the code in MATLAB (Figure 17).

```
[plot shape: step function, maximum: 0.004, minimum: -0.0004,
  pivot points: (0.002 seconds, 0.004), direction:
  ascending, x-intercepts: none, y-intercepts: (0 seconds,
  -0.0004)]
```

Listing 16. ChatGPT 4 response to the plot of code example 3 (Figure 17).

### 3.5.4 Example 4 of plant controller MATLAB code

MATLAB code for the most complex control system example (Listing 17). This is used for comparing two different control schemes (with switching and without) and plots a different graph depending on which variable is used (plot\_vector\_1 or plot\_vector\_2).

```
1 // Description of the plant system. Plant parameters are
   defined.
2 clc;
3 J = 3.2284E-6;
4 b = 3.5077E-6;
5 K = 0.0274;
6 R = 4;
7 L = 2.75E-6;
8 A = [0 1 0
9      0 -b/J K/J
10     0 -K/L -R/L];
11 B = [0 ; 0 ; 1/L];
12 C = [1 0 0];
13 D = [0];
14 plant = ss(A, B, C, D);
15
16
17 // Setting the sampling period
18 Ts=0.005;
19
20 // State space equation of plant-- discrete time
21 state_noise = 0;
22 ZOH = c2d(plant,Ts,'zoh');
23 [Apd, Bpd, Cp, Dpd] = ssdata(ZOH);
24 [K]=place(Apd,Bpd,[0.2,0.7,0.8]);
25
26 // Setting parameters for the simulation.
27
28 control_sampling_time=Ts;
```



```

29 simulation_sampling_time=10^(-6);
30 total_running_time=0.5;
31 total_samples=ceil(total_running_time/simulation_sampling_time);
32 time_axis_1=zeros(1,total_samples);
33 time_axis_2=zeros(1,total_samples);
34 plot_vector_1=zeros(1,total_samples);
35 plot_vector_2=zeros(1,total_samples);
36
37 // Defining the closed-loop plant control system
38 [dim_s,dim_i]=size(Bpd);
39 Ac=[Apd Bpd;-K zeros(dim_i,dim_i)];
40 Ao=[Apd Bpd;zeros(dim_i,dim_s) eye(dim_i)];
41 abs(eig(Ac));
42
43 // Setting some control execution patterns
44
45 Plant_1_sequence_2='1010010010101001001001010';
46 Plant_1_sequence_1='1011011010101011011001010';
47
48 // Initializing system variables and simulation environment
49
50 initial_value=[0.3;0.5;0.1];
51 [u_length]=size(Plant_1_sequence_1);
52 pointer=0;
53 sequence=Plant_1_sequence_1;
54 v=initial_value;
55 v_read=v;
56 u=0.01;
57
58 // Simulation of the plant control systems
59
60 for i=1:total_samples
61     dv=(A*v+B*(u)+state_noise)*simulation_sampling_time;
62     v=v+dv;
63     if(~mod(i,(control_sampling_time/simulation_sampling_time)))
64         u=(-1)*K*(v_read);
65         if((sequence(pointer+1)-48)==0)
66             v_read=v;
67         end
68         pointer=pointer+1;
69         pointer=mod(pointer,length);
70     end
71     if(norm(v)>1)
72         sequence=Plant_1_sequence_1;
73     end
74     time_axis_1(i)=total_running_time*i/total_samples;
75     plot_vector_1(i)=C*v;
76 end
77

```

```

78 pointer=0;
79 sequence=Plant_1_sequence_2;
80 v=initial_value;
81 v_read=v;
82 u=0.01;
83 for i=1:total_samples
84     dv=(A*v+B*(u)+state_noise)*simulation_sampling_time;
85     v=v+dv;
86     if(~mod(i,(control_sampling_time/simulation_sampling_time)))
87         u=(-1)*K*(v_read);
88         if((sequence(pointer+1)-48)==0)
89             v_read=v;
90         end
91         pointer=pointer+1;
92         pointer=mod(pointer,length);
93     end
94     if(norm(v)>1)
95         sequence=Plant_1_sequence_2;
96     end
97     time_axis_2(i)=total_running_time*i/total_samples;
98     plot_vector_2(i)=C*v;
99 end
100
101 plot(time_axis_1,plot_vector_1),xlabel('Time(s)'),
102 ylabel('Voltage'),title('Plant 1: with switching');
103 grid on;

```

Listing 17. Example 4 of plant controller MATLAB code (plot in Figure 18).

And ChatGPT 3.5's full response (Listing 18).

#### Step 1:####

The code initializes several variables related to a control system, such as the system matrices A, B, C, and D, the plant model, the sampling time, and the control gains. It then sets up parameters for simulation time, initializes arrays for time, plot vectors, and sequences. It also defines two different sequences for the plant, initializes the initial state of the system, and sets a state noise value.

#### Step 2:####

The initial and final values for the plot axes are not explicitly defined in the code. However, based on the simulation setup, the x-axis (time) ranges from 0 to 0.5 seconds, and the y-axis (voltage) values are determined by

```

the system dynamics during the simulation .

Step 3:####
This is a discrete-time closed-loop control system due to the
discretization of the plant using zero-order hold (ZOH)
and the use of a feedback controller to update the control
input based on the system state .

Step 4:####
The code simulates the system dynamics for two different
plant sequences with switching behavior. It calculates the
state evolution based on the system dynamics equation ,
updates the control input using a state feedback
controller at specific intervals , and switches between
sequences based on a condition related to the system
state 's norm exceeding a threshold .

Step 5:####
The plot will show the response of the system (voltage) over
time for Plant 1 with switching behavior. The plot will
exhibit changes in the voltage output based on the two
different plant sequences and the corresponding control
inputs. The plot will have multiple peaks and valleys ,
reflecting the dynamic behavior of the system under the
influence of control action and sequence switching .

Response to user:####
[plot shape: oscillatory with peaks and valleys , maximum:
<maximum voltage value> , minimum: <minimum voltage value> ,
pivot points: multiple peaks and valleys , direction:
mixed , x-intercepts: points where the plot intercepts the
x-axis , y-intercepts: points where the plot intercepts the
y-axis]

```

Listing 18. Example ChatGPT 3.5 response to code example 4 of plant controller MATLAB code (Listing 17, Figure 18)

And here is Grimoire's [CSY<sup>+</sup>24] response (Listing 19) to the actual plot of the code in MATLAB (Figure 18).

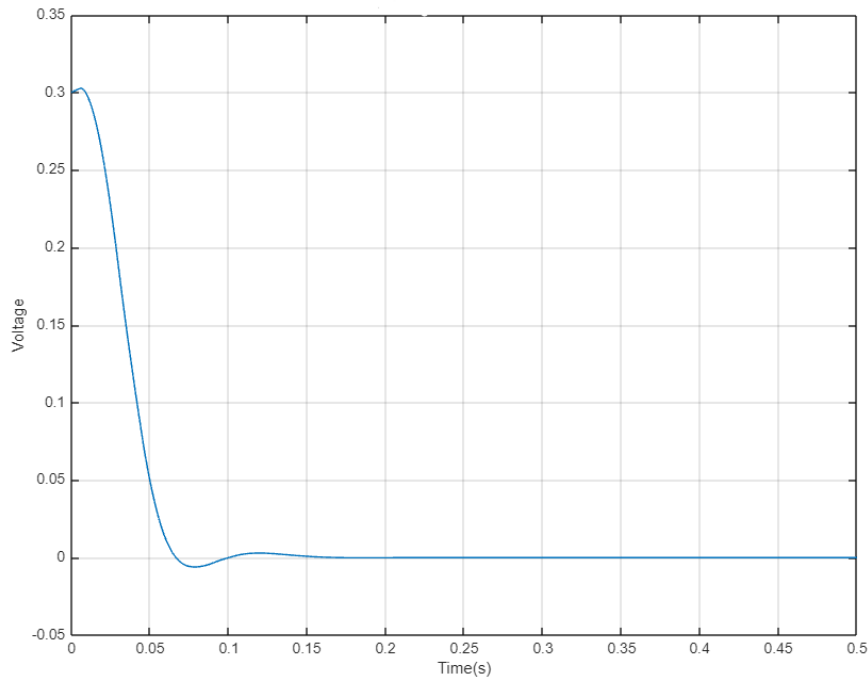


Figure 18. Plot for code example 4 (Listing 17).

```
[plot shape: exponential decay, maximum: approximately 0.35,
  minimum: approximately 0.0, pivot points: (0.05, 0.15),
  (0.05, -0.05), direction: descending, x-intercepts:
  approximately 0.05, y-intercepts: approximately 0.35]
```

Listing 19. ChatGPT 4 response to the plot of code example 4 (Figure 18).

### 3.5.5 Example 5 of plant controller Python code

Added one control code script in Python (Listing 20) to see how much the programming language matters.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
```

```

4 initial_loc=[[2],[3]]
5
6 a=np.array([[1,0.13],[0,1]])
7 b=np.array([[0.02559],[0.3937]])
8 c=np.array([1,0])
9 d=0
10
11 x=np.array(initial_loc)
12 x_prev=np.array([0],[0])
13 y=[]
14
15 x1=[]
16 x2=[]
17 u=[]
18
19 for t in range(150):
20     ut=np.dot(np.array([0.2935,0.4403]),x_prev)[0]
21     u.append(ut)
22
23     x_new=np.dot(a,x)+ut*b
24     x1.append(x_new[0][0])
25     x2.append(x_new[1][0])
26     y.append(np.dot(c,x_new)[0]+d*ut)
27
28     x_prev=x
29     x=x_new
30
31 plt.xlabel("t")
32 plt.plot(range(150),[x2[i]/x1[i] for i in range(150)])
33 plt.ylabel('x2/x1')
34 plt.show()

```

Listing 20. Example 5 of plant controller Python code.

And ChatGPT 3.5's full response (Listing 21).

```

Step 1:####
The code imports necessary libraries, initializes some
variables, defines matrices a, b, c, and d, sets initial
values for x, x_prev, and y, and then iterates over a loop
to calculate and store values in lists x1, x2, and u.
Finally, it plots x2/x1 against time.

Step 2:####
The initial values on the plot axes are set by the
initial_loc variable, which is [[2],[3]]. The final values

```

```

    will depend on the calculated values in the loop for x1
    and x2.

Step 3:####
This code represents a continuous, time-invariant, linear,
    SISO (Single Input Single Output) control system.

Step 4:####
- In each iteration, the code calculates the control input ut
    using the previous state x_prev and a predefined vector.
- It then computes the new state x_new based on the current
    state x, the control input ut, and matrices a and b.
- The code also calculates the output y based on the state
    x_new and matrices c and d.
- The values for x1, x2, and u are stored in lists x1, x2,
    and u respectively.
- The state x is updated for the next iteration.
- The code plots x2/x1 against time.

Step 5:####
The plot will show the ratio x2/x1 as a function of time. It
    will provide insights into the relationship between the
    two states x2 and x1 over the duration of the simulation.

Response to user:####
[plot shape: line, maximum: <value>, minimum: <value>, pivot
    points: N/A, direction: mixed, x-intercepts: N/A,
    y-intercepts: N/A]

```

Listing 21. Example ChatGPT 3.5 response to code example 5 of plant controller code (Listing 20, Figure 19)

And here is Grimoire's [CSY<sup>+</sup>24] response (Listing 22) to the actual plot of the code in MATLAB (Figure 19).

```

[plot shape: Sigmoidal, maximum: 1.450, minimum: 1.250, pivot
    points: t = 20, direction: Ascending, x-intercepts: None,
    y-intercepts: 1.250]

```

Listing 22. ChatGPT 4 response to the plot of code example 5 (Figure 19).

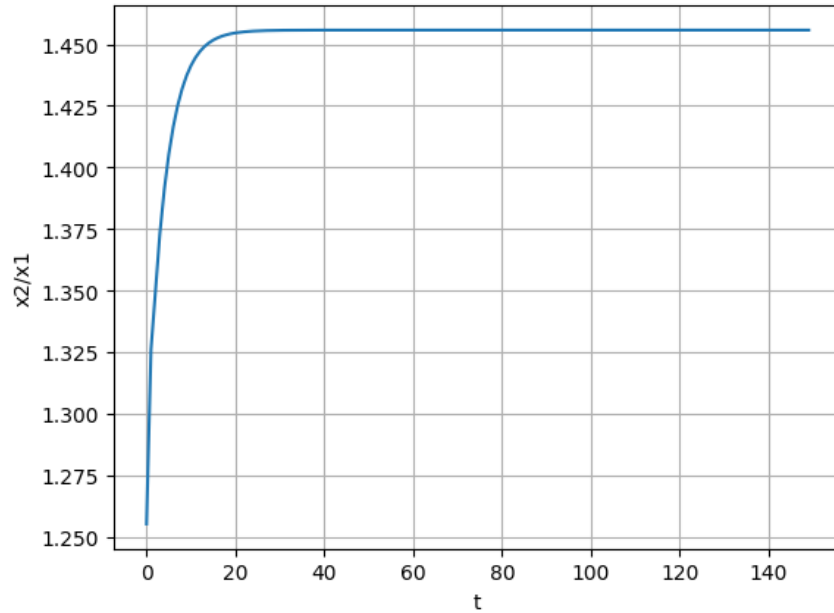


Figure 19. Plot for code example 5 (Listing 20).

## 4 Inference

While running the experiments, I noticed that ChatGPT 3.5 is pretty good at high level understanding but quite bad with details, especially when iterations and math are concerned. This is most likely because it doesn't know what math is, it only has some approximations of how numbers are often written in different contexts (a kind of a language perhaps).

### 4.1 Understanding the simplest code

After noticing that iterating code really confuses ChatGPT 3.5, we wanted to see how bad it really is so we ran a few tests. It started with a very simple prompt (Listing 23) and code snippet (Listing 24) which it managed to get everything correctly (Listing 25).

- 1 The user query is a piece of code that simulates and plots something.
- 2 Your task is to run the code and return the values for the x and y axes of the plot.
- 3 The final answer should contain only the values of the plot vectors.

```

4 The user query will be delimited with four hashtags ,
5 i.e. {delimiter}.
6
7 Step 1:{delimiter} Explain the code line by line.
8 Step 2:{delimiter} Figure out which variables are used for
  the plot.
9 Step 3:{delimiter} Emulate the code and display the values of
  the variables after the execution.
10
11 Use the following format:
12 Step 1:{delimiter} <step 1 reasoning>
13 Step 2:{delimiter} <step 2 reasoning>
14 Step 3:{delimiter} <step 3 reasoning>
15 Response to user:{delimiter} <response to customer>
16
17 Make sure to include {delimiter} to separate every step.

```

Listing 23. Simple code execution prompt.

```

1 a = []
2 for i in range(5):
3     a.append(i)

```

Listing 24. Simple loop example.

The high level description, the values and the plot - everything perfect.

```

1 Step 1:####
2 The code initializes an empty list named "a". It then
  iterates over a range of 5 values (0 to 4) using a for
  loop. During each iteration , the value of "i" is appended
  to the list "a".
3
4 Step 2:####
5 The variable "a" is used for the plot as it stores the values
  that will be plotted.
6
7 Step 3:####
8 After running the code, the list "a" will contain the values
  [0, 1, 2, 3, 4].

```



```

9
10 Response to user:####
11 The values for the x-axis would be [0, 1, 2, 3, 4] and the
    values for the y-axis would be [0, 1, 2, 3, 4].

```

Listing 25. Simple loop response.

Then we tried with a slightly more complex example where instead of simply appending to the list, there is some math involved with the loop index (Listing 26) and still no problemo (Listing 27).

```

1 a = []
2 for i in range(5):
3     a.append(2**i + 17)

```

Listing 26. Medium loop example.

```

1 ##### Step 1:
2 The code defines an empty list `a`. Then, it iterates over a
  range of 5 values, starting from 0 to 4, and appends the
  result of `2**i + 17` to the list `a`. Finally, it plots
  the values in list `a`.
3
4 ##### Step 2:
5 The variable `a` is used for plotting.
6
7 ##### Step 3:
8 After executing the code, the values of the variable `a` are:
9 a = [18, 19, 21, 25, 33]
10
11 Response to user:#### The values for the x-axis would be [0,
    1, 2, 3, 4] and the values for the y-axis would be [18,
    19, 21, 25, 33].

```

Listing 27. Medium loop response.

However when the math gets even more complicated and multiple variables have to be tracked in different directions (Listing 28), then problems start to arise.

```

1 a = []
2 b = 3
3 c = 8
4
5 for i in range(5):
6     a.append((2**i + b) / c)
7     b += 3
8     c -= 2

```

Listing 28. Hard loop example.

It still understands what the code does on a high level but iterating through it and executing the math is beyond it's capabilities (Listing 29). The code actually doesn't run: in the fifth iteration of the loop, the value of *c* will be 0 and a division by zero error will be thrown. And even if the code would not result in an error, the values are still wrong. The actual values in the list should be  $[0.5, 1.333..., 3.25, 10, NaN]$  (last element is *NaN* because of division by zero).

```

1 ##### Step 1:
2 Step 1:####
3 - We initialize an empty list `a`.
4 - We set `b` to 3.
5 - We set `c` to 8.
6 - We enter a loop that iterates 5 times.
7 - In each iteration, we calculate  $(2^i + b) / c$  and append it
  to list `a`.
8 - We increment `b` by 3 in each iteration.
9 - We decrement `c` by 2 in each iteration.
10 - Finally, we plot the values in list `a`.
11
12 Step 2:####
13 The variable used for the plot is `a`, which stores the
  calculated values.
14
15 Step 3:####
16 After emulating the code, the values of `a` for the plot are:
17 [0.625, 1.25, 0.875, 1.625, 2.125]
18
19 Response to user:####
20 The values for the x-axis are [0, 1, 2, 3, 4], and the values
  for the y-axis are [0.625, 1.25, 0.875, 1.625, 2.125].

```

Listing 29. Hard loop response.

## 5 Conclusion and Future Work

This thesis proposes a novel prompt engineering method to tune LLMs for solving the code comprehension task. We tune commercially available GPT 3.5 and GPT-4 to comprehend the Matlab script that simulates the behavior of control systems. Different types of control are studied, ranging from simple single-plant control to coupled control systems. Three types of prompts are designed and used to test the capacity of Matlab code comprehension. While the simple text prompt works well when the Matlab code simulates the behavior of a simple plant control system, more sophisticated logical prompts and numerical prompts are necessary to tune the LLM for comprehending complex Matlab scripts. Our Matlab code is not well commented and documented. Moreover, Matlab mainly solves the numerical time-invariant state-space equations under the hood while simulating the behavior of the control system. This is very different from the training methods of LLMs, which are trained in an auto-regressive way on the text corpus. The large deviations of the text distributions on which the LLMs are trained and the coefficients matrix in the state-space equations make code comprehension a hard problem. This thesis proposes a new concept of numerical prompt to address this problem. This thesis proposes a new metric as an evaluation measure to check the correctness of understanding of LLMs. The new metric is generalizable and can be used in tasks beyond code comprehension. Our research findings hint current commercial LLMs (GPT 3.5 and GPT-4) are good for solving language tasks but are not yet optimal for solving complex tasks of code comprehension.

In our future work, we will extend our works of numerical prompts for code comprehension tasks on stochastic control systems. We will refine our proposed new metric to other classes of problems.

Also it may be worthwhile to test these method on some open source models where we could get more insight into what the model is actually doing.

## References

- [CHW24] Daixuan Cheng, Shaohan Huang, and Furu Wei. Adapting large language models via reading comprehension, 2024.
- [CSY<sup>+</sup>24] Ding Chen, Shichao Song, Qingchen Yu, Zhiyu Li, Wenjin Wang, Feiyu Xiong, and Bo Tang. Grimoire is all you need for enhancing large language models. *arXiv preprint arXiv:2401.03385*, 2024.
- [CTJ<sup>+</sup>21] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.
- [FGM<sup>+</sup>23] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. Automated repair of programs from large language models, 2023.
- [HBK<sup>+</sup>21] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps, 2021.
- [Hos22] E. Hossain. *MATLAB and Simulink Crash Course for Engineers*. Springer International Publishing, 2022.
- [LCC<sup>+</sup>22] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and

- Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, December 2022.
- [NMBM98] D. Nesic, Iven Mareels, Georges Bastin, and Robert Mahony. Output dead beat control for a class of planar polynomial systems. *SIAM Journal on Control and Optimization*, 36:253–272, 01 1998.
- [Pal22] Sankaran Palani. *Automatic Control Systems With MATLAB Second Edition*. ANE Books India, 2022.
- [WFH<sup>+</sup>23] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt, 2023.
- [YLZ<sup>+</sup>23] Zhiqiang Yuan, Junwei Liu, Qiancheng Zi, Mingwei Liu, Xin Peng, and Yiling Lou. Evaluating instruction-tuned large language models on code comprehension and generation, 2023.

# **Appendix**

## **I. Code repository**

The code that was used for this thesis, is available in this Colab document:

<https://colab.research.google.com/drive/1HwaxiWp7EbC2dru3KWtfzPmHBST1puzy>

## II. Licence

### Non-exclusive licence to reproduce thesis and make thesis public

I, **Sander Sats**,

(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

**Control Code Comprehension with Large Language Models (LLMs),**

(title of thesis)

supervised by Kallol Roy.

(supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Sander Sats

**15/05/2024**