Applying LLMs to Alloy models*

*CS846 project report

Mathew Kuthur James
David Cheriton School of Computer Science
Univeristy of Waterloo
Waterloo, Canada
m2kuthur@uwaterloo.ca

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert.**

I. Introduction

A. Declarative Modeling and Alloy

Software modeling is an important part of software development, used for managing complexity and ensuring correctness [cite].

Alloy is a formal declarative specification language, for expressing structural constraints and behaviour.

The following example is from Jackson's Software Abstractions, with added annotations:

```
module language/grandpa1 ---- Page 84, 85

// a sig declares a set of atoms, with shared
    properties
abstract sig Person {
    father: lone Man, // lone is a condition on
        multiplicity, this means that every
        Person p has exactly one Man m as p.
        father
    mother: lone Woman
    }

sig Man extends Person {
    // Man inherits all the properties of Person
        , with additional properties
    wife: lone Woman
    }

sig Woman extends Person {
    husband: lone Man
    }

fact {
    // a fact is a condition on the possible
        relations between atoms
    no p: Person | p in p.^(mother+father)
    wife = ~husband
    }

assert NoSelfFather {
    // assertions are claims about the relations
         between atoms, which are not
        necessarily true. They may be true as a
        consequence of the definitions of the
        sigs and facts, which is determined by
        the Analyzer
    no m: Man | m = m.father
    }

// This should not find any counterexample.
check NoSelfFather

fun grandpas [p: Person] : set Person {
    // fun f [a: T]: U takes an argument of type
         T and returns something of type U
    p.(mother+father).father
    }

pred ownGrandpa [p: Person] {
    // a function that returns a boolean value
    p in p.grandpas
    }

// This should not find any instance.
run ownGrandpa for 4 Person

assert NoSelfGrandpa {
    no p: Person | p in p.grandpas
    }

// This should not find any counterexample
check NoSelfGrandpa for 4 Person
```

This is a simple model of a familial structure, which asserts that no `Person` p can be their own grandfather or their own father.

Properties of more complex systems can be modelled similarly, after which the Alloy Analyzer is used to look for counterexamples to assertions about the model's properties.

B. LLMs and Deepseek

A large language model (LLM) is a machine learning model designed for natural language processing, such as language and code generation. LLMs are trained on a vast corpus of existing examples of text, and can be used to produce new examples of the text. When combined with a prompt, the LLM follows instructions in a manner similar to a chatbot.

Mixture of Experts (MoE) is a machine learning technique where learners operate on the same input, with their outputs combined by a weighting function, to produce a single output.

DeepSeek is a MoE LLM released as an open-weight model. An open-weight model can be run by anyone on their own

hardware, as opposed to closed-weight LLMs like Chat-GPT, which can be accesed only via an API.

The research questions of this project are:

- Given an Alloy model, how well can deepseek summarize the model in plain english?
- Can deepseek write valid Alloy models?
- Given a summary of a model in plain english, how well can deepseek capture the requirements when it constructs the model in Alloy?

The results shed light on LLMs' ability to comprehend and use declarative modelling languages, which could increase the adoption of software modelling in the industry. Since a significant hurdle to the adoption of tools such as Alloy in production is the unfamiliarity of the developers with declarative modelling, the introduction of AI-assisted creation of models may increase the use of Alloy and similar tools (TLA+, Z, B etc.) in software development.

When the earlier example model is fed into deepseek with a request for a summary, the following is produced:

```
    This Alloy module defines a family relationship
        structure and enforces constraints to
        prevent impossible genealogical cycles (e.g
        ., someone being their own grandfather).

    Key Components

    Signatures (sig)

        abstract sig Person: Base type with optional
             father (Man) and mother (Woman).

        sig Man extends Person: Adds wife (lone
            Woman).

        sig Woman extends Person: Adds husband (lone
             Man).

        Constraints (fact)

        No cycles: no p: Person | p in p.^(mother+
            father) (no person is their own ancestor
            ).

        Spouse symmetry: wife = ~husband (wife/
            husband are mutual).

    Functions & Predicates

        grandpas[p: Person]: Returns a person's
            grandfathers (mother/father -> father).

        ownGrandpa[p: Person]: Checks if someone is
            their own grandpa (impossible by design)
            .

        Verification

        Assertions:

            NoSelfFather: No man is his own father.

            NoSelfGrandpa: No person is their own
                grandpa.

        Checks:

            check NoSelfFather / check NoSelfGrandpa
                : Verify assertions hold.

            run ownGrandpa for 4: Confirms no valid
                instance exists.

    Purpo
    se

        Model family trees without contradictions.

        Ensure logical consistency (e.g., no time-
            traveling grandparents).

    The Analyzer confirms all constraints are satisfied.

    Want to extend this? For example, add siblings or
        marriages between cousins?
```

II. Methodology

A. Data Collection

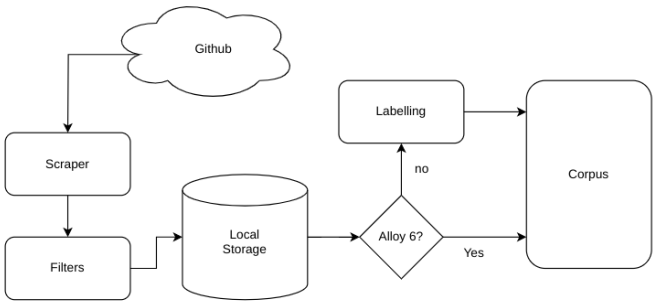

Fig. 1. Data collection.

The primary data source for Alloy models is GitHub, whose public repositories with `.als` files are scraped by a script. After scraping, the files are passed through a filter which determines the file type, before moving the files to local storage.

Given a corpus of `.als` files containing Alloy code, the version of Alloy is determined via a simple check of the keywords used. Alloy 6 is not back-compatible with earlier versions of Alloy, and is characterized by the following keywords (which do not appear in Alloy 5 code):

```
    keywords = "after always before enabled event
        eventually historically invariant modifies
        once releases since steps triggered until
        var".split(" ")
```

Determining the version is carried out by a simple algorithm shown below:

```
    def remove_strings(text):
        text = remove_escape_sq(text)
        text = re.sub(r"\"[^\"]*\"","",text,re.
            DOTALL)
        return text

def check_presence(text):
        text = remove_strings(text)
        text.replace("\n"," ")
        text.replace("\t"," ")
```

```
        text.replace("\r"," ")
        for k in keywords:
                if " "+k+" " in text:
                        return True
        return False
```

Alloy strings are enclosed in double quotes. Before checking for the presence of keywords, the string literals are eliminated via a regular expression, since a simple word search will match all occurances of the Alloy-6 keywords, even if they are part of a string literal (which is semantically distinct from occurance in the code). The Alloy version is inserted as a comment at the top of the model code, as:

```
    // This is an Alloy 6 model
```

For models whose version is confirmed to be Alloy 6. No modification is done to models which could be interpreted as either Alloy 5 or Alloy 6.

B. Invoking the LLM

Engineering a pipeline to extract data automatically from LLMs has the following requirements:

- Stability - if a process crashes, the data loss should be minimal.
- Parallelizability - the pipeline should be easy to expand to run on multiple machines, without relying on interprocess communication between machines.
- Reproducibility - Since LLMs are stochastic, the results produced are pseudorandom. For independent verification, the results must be reproducible.

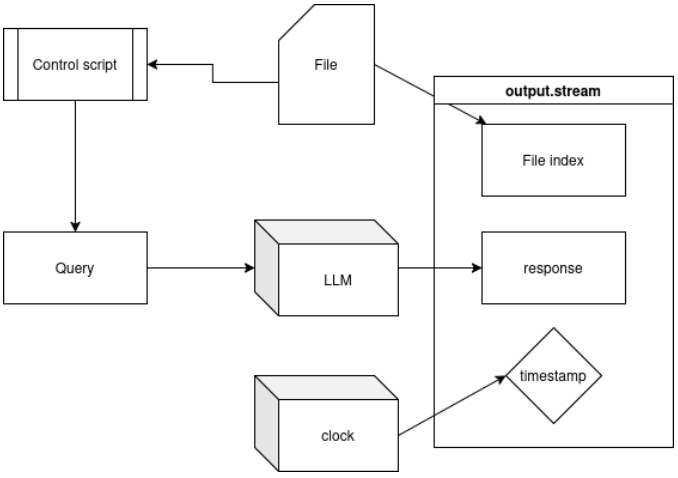

Fig. 2. LLM control process.

The control script reads a file, constructs the prompt from the file, and feeds it into the LLM. The seed is set to be 0, for reproducibility. The output is stored in a `.stream` file, which consists of the following metadata in addition to the response:

- The generation timestamp (unix epoch)
- An index which points to the file from which the prompt was generated

After each response generation, a `.stream` file is created and stored in permanent storage. This ensures that if the process crashes:

- All data stored thus far is saved (except the response being generated during the crash).
- The process can be recovered by reading the indices of the generating files in the `.stream` file, which allows it to continue with unprocessed files.
- A real-time alert can be generated for crashes by a separate process that continuously scans the timestamps of the existing `.stream` files and raising an alert if there is a large difference between the current time and the timestamp of the last generated file.

C. Prompt Engineering

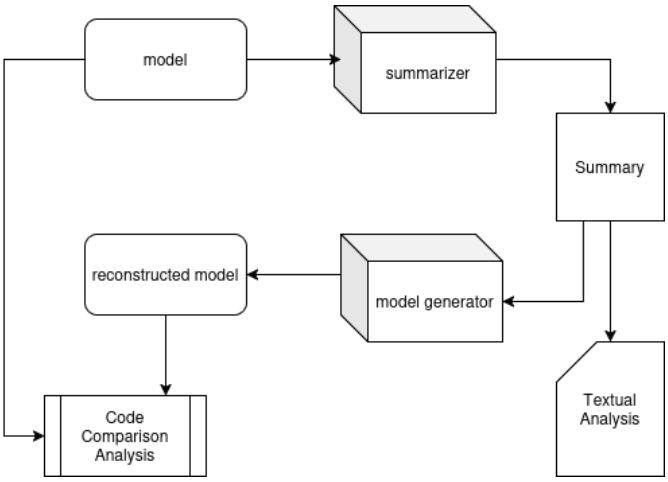

Fig. 3. LLM pipeline.

In accordance with the standard best pracitces for prompt engineering the instructions were appended to the beginning of the prompt, to ensure that sufficient attention was allocated to the instruction.

The context was clearly separated from the instructions, so the line between the where the instructions to the LLM ends and the model to analyze begins is clear. The two prepended instructions are:

```
    "This is sample Alloy code. Provide an accurate
        summary of the code, make it as detailed as
        possible\n"+summary
```

```
    "Here is a summary of an Alloy model in plain
        english. Generate the Alloy code:"+summary
        +"\n Provide only Alloy code, not a summary"
```

For the second experiment (the generation of the code from the prompt), about 91% of the queries resulted in Alloy code. The presence of code in the response was identified by looking for the presence of triple quotes in the response, which is used to indicate code segments.

The general structure of a response by deepseek is:

```
<think>
(internal thoughts)
</think>
(explanation)
'''
(code)
'''
(summary of explanation)
```

The constituent parts of the response a separated out via matching using regular expressions.

III. Results

A. Analysis of the Corpus

Initial analysis of the corpus consists of examining simple characteristics like the number of lines, number of tokens and number of characters.

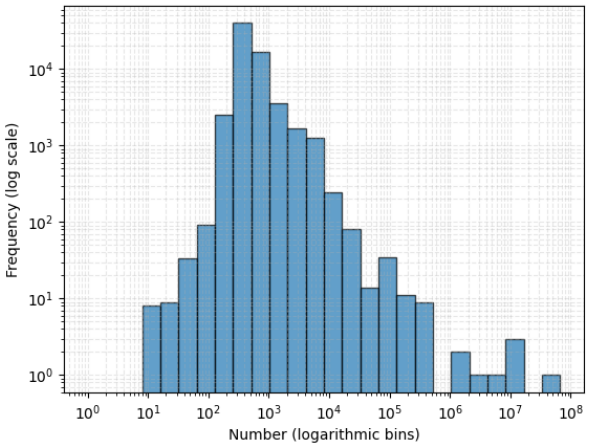

Fig. 4. Number of characters, distribution in corpus.

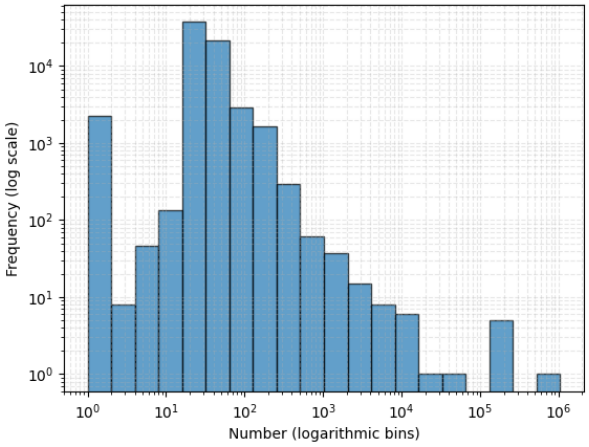

Fig. 5. Number of lines, distribution in corpus.

When examining the number of lines, there seems to be a large number of `.als` files with very few lines (which could be because of a large number of empty or very small models in the corpus).

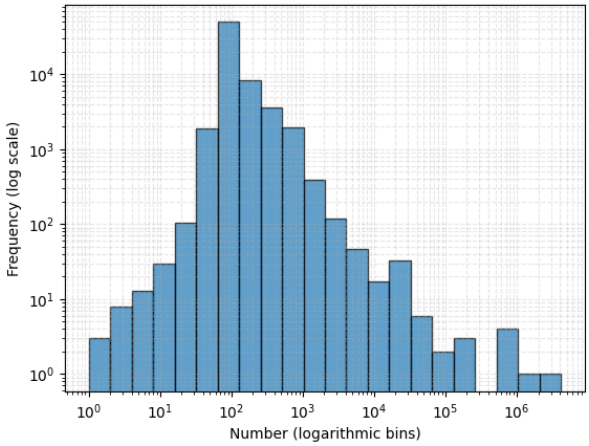

Fig. 6. Number of tokens, distribution in corpus.

A similar analysis is carried out for the number of words and sentences in the summaries generated by deepseek. The data is split based on the actual summary and the 'think' section, which shows deepseek's reasoning.

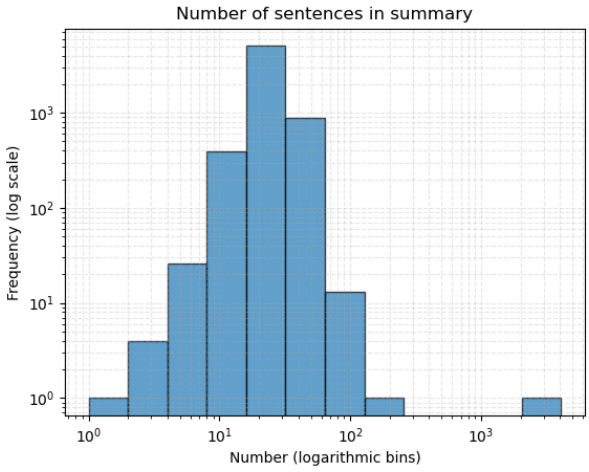

Fig. 7. Number of sentences in summary.

The relative sharpness of the distribution peaks for the responses, in comparison to the distributions for the original model, show that deepseek tends to have a fairly limited response window size.

Further investigation of the relationship between the size of the model being summarized and the size of the summary is shown below:

From the data, it is clear that the LLM is capable of producing models whose length is at or below the mode for the dataset. However, for large models, the reconstructions produced by the LLM rarely reach a similar length. This

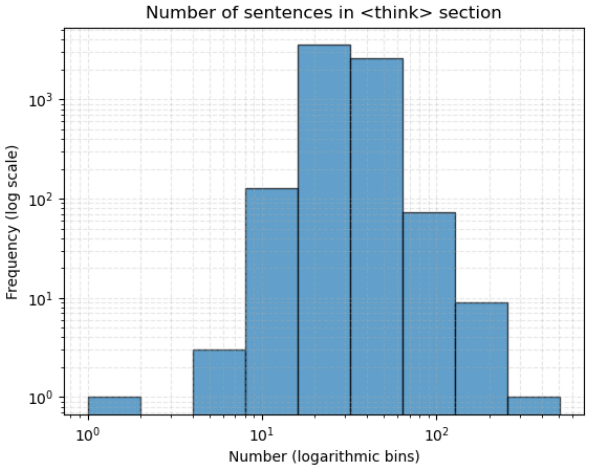

Fig. 8. Number of sentences in ¡think¿ section.

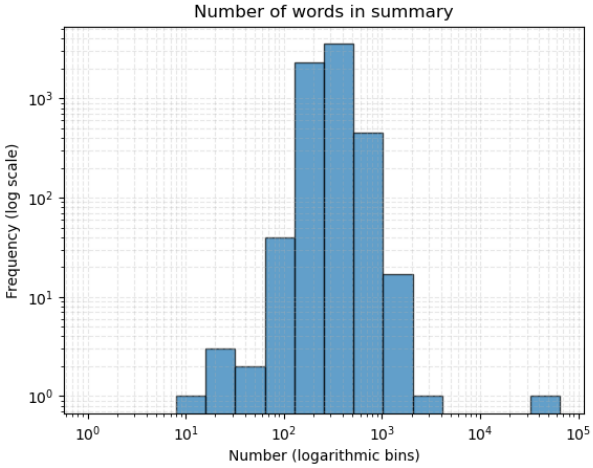

Fig. 9. Number of words in summary.

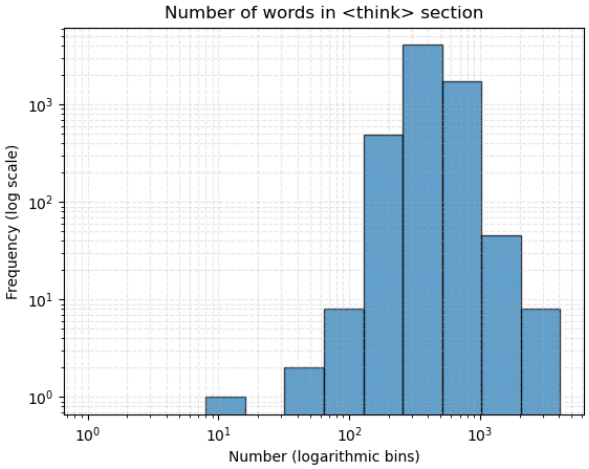

Fig. 10. Number of words in ¡think¿ section.

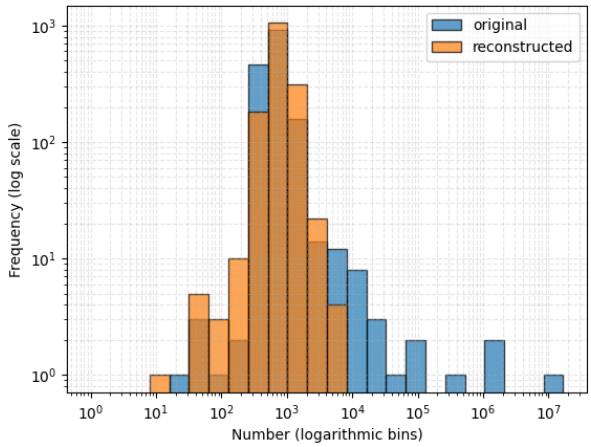

Fig. 11. Comparing number of characters in the corpus vs the reconstructed models.

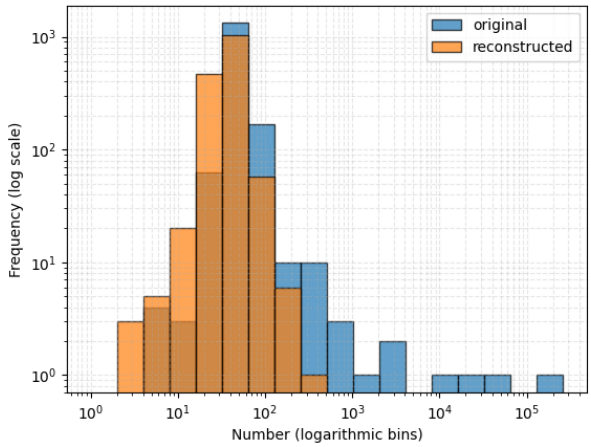

Fig. 12. Comparing number of lines in the corpus vs the reconstructed models.

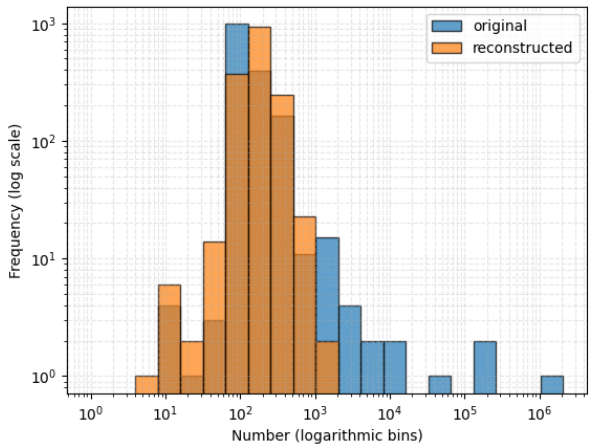

Fig. 13. Comparing number of tokens in the corpus vs the reconstructed models.

suggests that the best use of AI involves the creation of small modular Alloy code, rather than large monoliths.

The difference in distributions at the left tail indicates that the LLM is biased towards generating smaller models than the ones in the corpus. This suggests that in the process of summarization and reconstruction, there is either a decrease in verbosity, or a loss of detail and complexity.

When the relation between the sizes of the original model, the reconstruction and the summary is explored in further detail, the following becomes apparent:

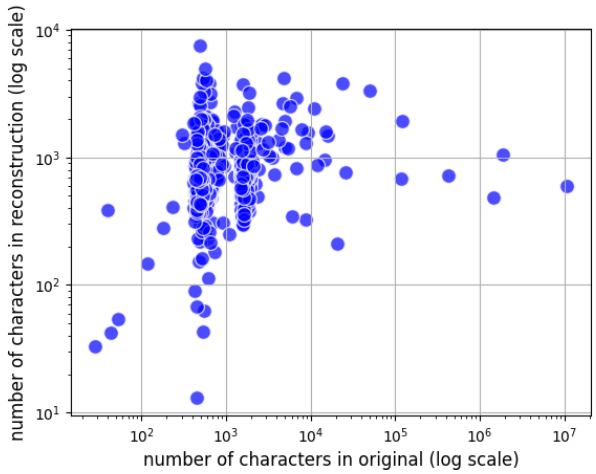

Fig. 14. correlation between size of original and reconstructed model.

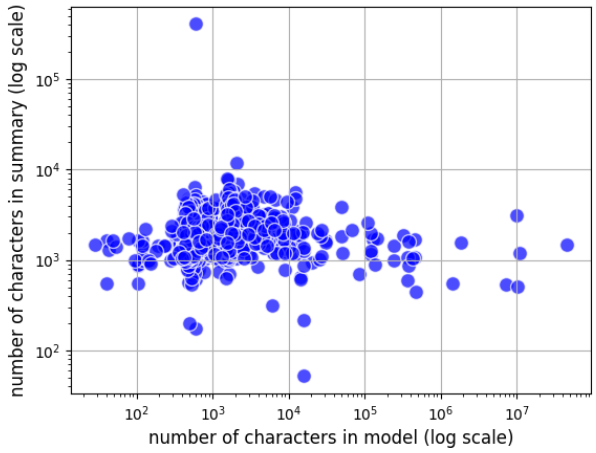

Fig. 15. correlation between size of model and LLM-generated summary.

- The reconstructed model shows a strong positive correlation upto models of around 1000 lines, after which the size of the reconstructed model plateaus out. This strongly suggests that the LLM tends to recreates models of a similar size as the original, provided it is not hampered by the response window.
- There is no correlation between the length of the summary and the length of the model being summarized, possibly because the LLM uses all the available response length to explain the model. In case of simple models, the explanation could be more verbose, expanding to fill up the response window.

B. Analysis of the Comments

For languages which are designed to be easily readable, the best coding standards recommend self-commented code whose purpose is evident upon reading (for instance, using practices like descriptive variable names). In case the language's syntax is highly specialized or otherwise difficult to read for the layperson, a liberal policy is recommended for writing comments (since the difficulties cuased by the presence of an unnecessary comment is much less than the absence of a necessary comment).

In general, we expect the corpus to have fewer comments than would be seen in typical production Alloy code in a professional environment, since the corpus includes Alloy code written as part of assignments and small personal projects. When the commenting habits are compared between the corpus and the reconstructed code:

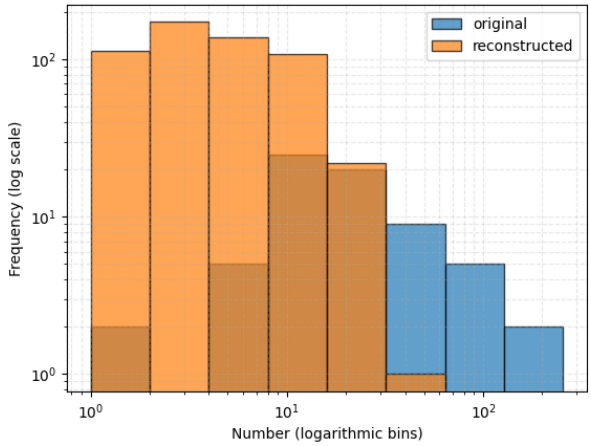

Fig. 16. Comparing number of comments in the corpus vs the reconstructed models.

The key conclusions are:

- LLMs tend to comment less when reconstructing a model from a summary, in comparison to the original models.
- The comments written by LLMs tend to be shorter than the comments in the original model, possibly because the explanation of each element is presented in a separate section outside the code.

The commenting behaviour of LLMs could be amended via amending the prompt to include phrases such as "Explain the code in comments, not in a separate section". LLMs which can reliably do this present a significant advantage to the user, who need not keep switching between the code and the explanation when reading the reconstructed model.

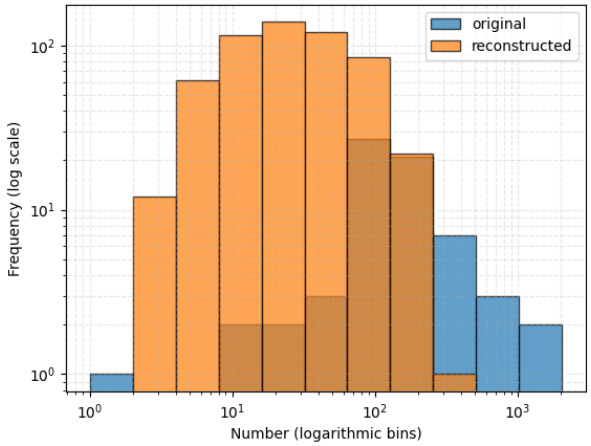

Fig. 17. Comparing total number of words in the comments in the corpus vs the reconstructed models.

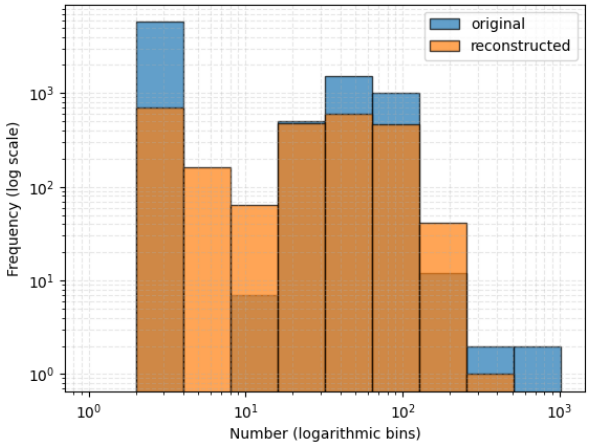

Fig. 19. Comparing size of the signature bodies in the corpus vs the reconstructed models.

C. Analysis of Signatures

Alloy models describe relations between atoms, whose types are described by signatures. Inheritance is applied to signatures using the `extends` keyword. For instance, `Man extends Person` and `Woman extends Person` in the earlier example show inheritance, where `Man` and `Woman` are subtypes of `Person`.

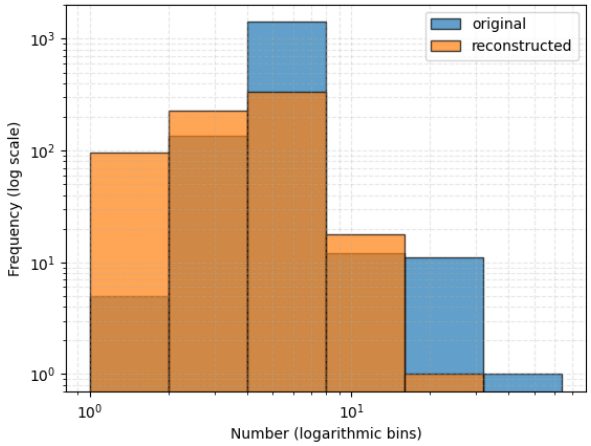

Fig. 18. Comparing number of signatures in the corpus vs the reconstructed models.

When comparing the signatures generated by the LLM with the original corpus:

The LLM tends to create models with under 10 signatures. This suggests that a low accuracy for the reconstruction of large models, since a reduction in the number of signaures tends to correspond to a reduction in the model's complexity. Poorly designed signature schema could be simplified by merging signatures and eliminating unnecessary abstraction (for instance, `Person` can be eliminated by duplicating its code in both `Man` and `Woman`, reducing the total number of signatures by 1), but such reductions often result in an increase in the sizes of the signatures thus produced, which is not the case as seen in the comparison of sizes, which reveals that the LLM is slightly biased towards producing signatures with smaller sizes.

This suggests that creating multiple `.als` files with a small number of small signatures will result in a better model, which is in line with generally recommended coding practices.

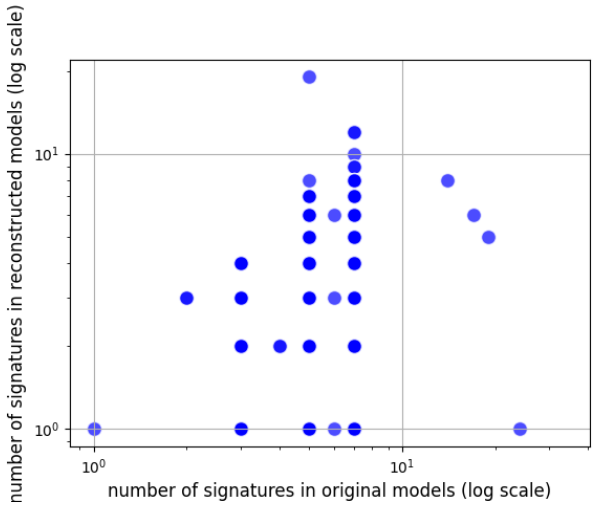

Fig. 20. Correlation between number of signatures in the original and reconstructed models.

This shows that the reconstructed models tends to have a number of signatures lesser than or equal to the number of signatures in the original model, with a quick drop-off for models having more than 10 signatures, which supports the conclusions drawn from the earlier results.

Similar analyses, when performed for other elements like facts, preds, asserts and funs, do not produce enough data to

draw any strong conclusions. The distributions are included in the appendix.

D. Analysis of Signature Inheritance

In general, inheritance can be represented via a directed acyclic graph [cite], where each node corresponds to a single signature. Signatures are categorized as follows:

- type signatures (declared using the `extends` keyword)
- subset signatures (declared using the `in` keyword)

Due to additional constraints imposed on type signatures in Alloy, the graph associated with the relations between them forms a forest-like structure, one with a set of rooted trees [cite] in the top-level types.

While determining isomorphism for arbitrary graphs is a hard problem, determining isomorphism for sets of rooted trees is simple.

Algorithm: Binary encoding

Require: $root$ points to the root node of tree
Ensure: s is the binary encoding of $root$

function encode($root$)
  if $root = nullptr$ **then**
    return ε
  end if
  $c \leftarrow [encode(c) : c \in root.children]$
  $s \leftarrow$ "0"
  Sort c lexicographically
  for $i \leftarrow 1$ **to** $c.length$ **do**
    $s \leftarrow s \cdot c[i]$
  end for
  $s \leftarrow s \cdot$ "1"
  return s
end function

The given algorithm produces a unique binary string for every rooted tree. For a forest of trees, a new tree is constructed where each top-level node in the forest is a child of a ultimate root node, to which this algorithm is then applied.

Algorithm: Tree decoding

Ensure: s is the binary encoding of a tree
Require: $root$ points to the root node of decoded tree

function decode(s)
  $root \leftarrow nullptr$
  for $i \leftarrow 1$ **to** $s.length$ **do**
    $n \leftarrow s[i]$
    if $n =$ "0" **then**
      $c \leftarrow Node()$
      $c.parent \leftarrow root$
      $root \leftarrow c$
    else
      $root \leftarrow root.parent$
    end if
  end for
  return $root$
end function

For brevity, a formal proof of correctness of the algorithms is omitted. The existence of a unique encoding and a reversible reconstruction function prove that the encoding is bijective. Applying this algorithm to the graphs associated with type signature inheritance relations, we used it along with a hashtable to enumerate the different inheritance relations in the corpus and identify the relative distributions.

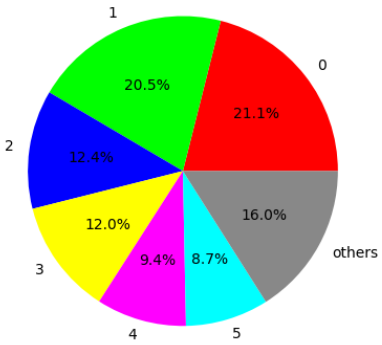

Fig. 21. distribution of inheritance hierarchies (original).

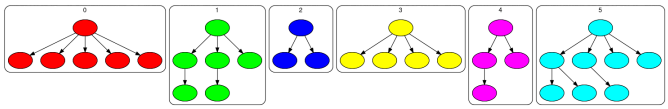

Fig. 22. Legend: Inheritance Structures (original).

The legend shows the graph representations of the inheritance structures, after the introduction of a root node for the forest. The original corpus consists of a large number of models with no ingeritance at all (as seen in the graphs with a depth of 1), with a few more complex structures occuring often (as seen in the graphs with a depth greater than 1).

In comparison, the models generated by the LLM consist mainly of simple inheritance structures, which suggests that the generated code often consists of signatures that do not inherit from other signatures. This could either be due to some inherent bias against inheritance when writing models, or due to the inheritance relations not being captured accurately in the summaries from which the models are reconstructed.

E. Manual Analysis of Summaries

The analysis of summaries of models was carried out manually after certain automatic filters:

1) Length: To ensure that the models can be understood by a manual reviewer, all models with more than a 100 lines are removed from the dataset. To eliminate trivial models, models with less than 10 lines are removed.

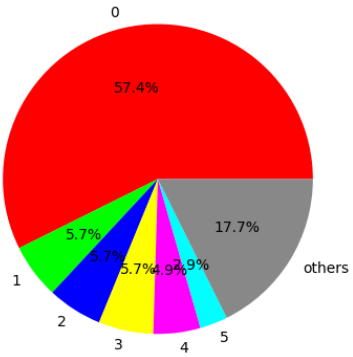

Fig. 23. distribution of inheritance hierarchies (reconstructed).

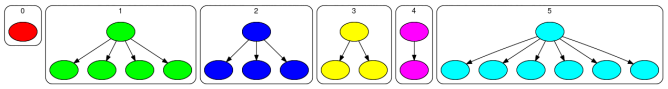

Fig. 24. Legend: Inheritance Structures (reconstructed).

2) Duplicate Data: To remove models which are very similar to each other, the following method was used to measure the degree of similarity between models:

Algorithm: Model Similarity

Require: $s1, s2$ (strings with spaces removed)
Ensure: $similarity(s1, s2)$

$N \leftarrow min(s1.length, s2.length)$
$ct \leftarrow 0$
for $i \leftarrow 1$ **to** N **do**
  if $x > 0$ **then**
    $ct \leftarrow ct + 1$
  end if
end for
return ct/N

For every pair of models $(m1, m2)$ such that the similarity score exceeds 0.5, one of the models is removed from the dataset. This process continues until no such pairs remain.

3) Language: Since the manual reviewer is familiar only with english, all models with non-ASCII characters were removed (since Alloy syntax and the english is captured completely by the ASCII character set). This was necessary since the language of the summary generated by the LLM is influenced by the language of the comments and keywords in the model text.

4) Composition: The manual reviewer went through the remianing models and eliminated those which were part of bigger projects, where identifiers not present in the text itself were referenced. This was done since such models rely on contextual information not available to the LLM, making the LLM summaries incomplete.

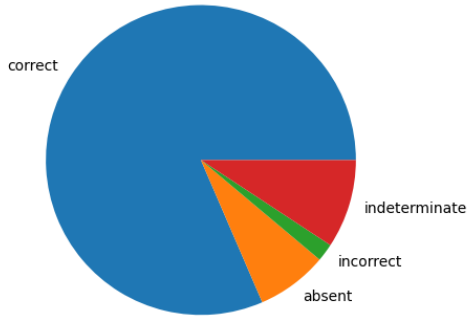

Fig. 25. Distribution of ratings from manual analysis.

After these filtration steps, a total of 50 models were chosen for manual review by the reviewer. From the 50 models, a reviewer counts the number of elements in each model (defined as either a `sig`, `fact`, `assert`, `pred` or a `fun`), and the number of elements correctly described in the summary.

IV. Limitations

A. Selection Bias

Manual analysis of summaries is carried out only for models which are easy to read. This biases it towards models of around 100 lines, written using english identifiers (which is the language the authors are familiar with).

Large models are left out of the manual analysis. Since the automatic analyses strongly suggest that the performance of the LLM is dependent on the size of the model, the results of the manual analysis cannot be extended to apply to large models.

Very short (or empty) models, with no more than 2 signatures, are often captured exactly by the summary, making manual analysis shed no new light on the performance. Furthermore, such models are unlikely to be used in production, so manual analysis does not result in useful conclusions about the application of LLMs in generating models in industrial software engineering.

B. Correctness of models

Automatic analyses only describe the coding styles of the LLM and compare it to the styles of the original model. It does not say anything about the correctness of the generated models, since it does not compare the ASTs generated by the Alloy Ananlyzer to check for semantic similarity.

The conclusions drawn from automatic analysis are generally superficial, and cannot be used to make strong conclusions about the LLM's overall effectiveness at model reconstruction.

C. Code analysis

The analysis is performed via an ad-hoc parser created for the purpose of the project, and not the official Alloy Analyzer, since there is a large section of generated code which is almost syntactically correct, from which producing the official AST is impossible without manual editing to fix syntax errors (a costly process). The custom parser imposes looser conditions on syntactic correctness, which allows it to analyse LLM-generated code which is not strictly compliant with Alloy standards.

Furthermore, only static analysis is performed, since dynamic analysis requires that the code be parsed and excuted by the Analyzer. This poses a significant hindrance to drawing conclusions about the correctness of the generated models.

D. Data duplication

When assignments are uploaded to GitHub (from which the original model is scraped), the models used in the assignments tend to appear several times in the dataset. This was accounted for in manual analysis (via rejecting multiple instances of similar-looking models, for diversity) but is a significant factor in the automatic analyses. One possible mitigation strategy is to use a distance metric between strings to identify groups of models that appear to originate from the same base model, for deduplication of data.

V. Future work

A possible avenue for future work could be a comparative study between different LLMs and their abilities to generate Alloy code. This could cover both general purpose LLMs (like GPT-4, Claude, Gemini etc.) and variants of LLMS specifically trained to develop code (like codellama). As LLMs get more powerful at writing code, we would expect to see an increase in the ability to generate valid and useful Alloy code.

Given a set of valid models, a possible extension of this project is to use the ASTs generated by Alloy Analyzer for structural analysis of the generated code. One can go further and perform dynamic code analysis by running the reconstructed models using Alloy Analyzer to see how closely the outputs align with the original models.

A.

VI. Introduction

This document is a model and instructions for LaTeX. Please observe the conference page limits. For more information about how to become an IEEE Conference author or how to write your paper, please visit IEEE Conference Author Center website: https://conferences.ieeeauthorcenter.ieee.org/.

A. Maintaining the Integrity of the Specifications

The IEEEtran class file is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

VII. Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections VII-A to VII-H below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not number text heads—LaTeX will do that for you.

A. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, ac, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as “3.5-inch disk drive”.
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: “Wb/m^2” or “webers per square meter”, not “webers/m^2”. Spell out units when they appear in text: “. . . a few henries”, not “. . . a few H”.
- Use a zero before decimal points: “0.25”, not “.25”. Use “cm^3”, not “cc”.)

C. Equations

Number equations consecutively. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{1}$$

Be sure that the symbols in your equation have been defined before or immediately following the equation. Use “(1)”, not “Eq. (1)” or “equation (1)”, except at the beginning of a sentence: “Equation (1) is . . .”

D. LaTeX-Specific Advice

Please use "soft" (e.g., `\eqref{Eq}`) cross references instead of "hard" references (e.g., `(1)`). That will make it possible to combine sections, add equations, or change the order of figures or citations without having to go through the file line by line.

Please don't use the `{eqnarray}` equation environment. Use `{align}` or `{IEEEeqnarray}` instead. The `{eqnarray}` environment leaves unsightly spaces around relation symbols.

Please note that the `{subequations}` environment in LaTeX will increment the main equation counter even when there are no equation numbers displayed. If you forget that, you might write an article in which the equation numbers skip from (17) to (20), causing the copy editors to wonder if you've discovered a new method of counting.

BibTeX does not work by magic. It doesn't get the bibliographic data from thin air but from .bib files. If you use BibTeX to produce a bibliography you must send the .bib files.

LaTeX can't read your mind. If you assign the same label to a subsubsection and a table, you might find that Table I has been cross referenced as Table IV-B3.

LaTeX does not have precognitive abilities. If you put a `\label` command before the command that updates the counter it's supposed to be using, the label will pick up the last counter to be cross referenced instead. In particular, a `\label` command should not go before the caption of a figure or a table.

Do not use `\nonumber` inside the `{array}` environment. It will not stop equation numbers inside `{array}` (there won't be any anyway) and it might stop a wanted equation number in the surrounding equation.

E. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [7].

F. Authors and Affiliations

The class file is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

G. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is "Heading 5". Use "figure caption" for your Figure captions, and "table head" for your table title. Run-in heads, such as "Abstract", will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced.

H. Figures and Tables

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. 25", even at the beginning of a sentence.

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an

TABLE I
TABLE TYPE STYLES

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
copy	More table copy[a]		

[a]Sample of a Table footnote.

example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.
[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
[4] K. Elissa, "Title of paper if known," unpublished.
[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.
[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
[8] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, arXiv:1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114
[9] S. Liu, "Wi-Fi Energy Detection Testbed (12MTC)," 2023, gitHub repository. [Online]. Available: https://github.com/liustone99/Wi-Fi-Energy-Detection-Testbed-12MTC
[10] "Treatment episode data set: discharges (TEDS-D): concatenated, 2006 to 2009." U.S. Department of Health and Human Services, Substance Abuse and Mental Health Services Administration, Office of Applied Studies, August, 2013, DOI:10.3886/ICPSR30122.v2
[11] K. Eves and J. Valasek, "Adaptive control for singularly perturbed systems examples," Code Ocean, Aug. 2023. [Online]. Available: https://codeocean.com/capsule/4989235/tree