SYNTHESIS OF MATHEMATICAL PROGRAMS FROM NATURAL LANGUAGE SPECIFICATIONS

ganesh.prasathr@tcs.com, shirish.karande@tcs.com

ABSTRACT

Several decision problems that are encountered in various business domains can be modeled as mathematical programs, i.e. optimization problems. The process of conducting such modeling often requires the involvement of experts trained in operations research and advanced algorithms. Surprisingly, despite the significant advances in the methods for program and code synthesis, AutoML, learning to optimize etc., there has been little or no attention paid to automating the task of synthesizing mathematical programs. We imagine a scenario where the specifications for modeling, i.e. the objective and constraints are expressed in an unstructured form in natural language (NL) and the mathematical program has to be synthesized from such an NL specification. In this work we evaluate the efficacy of employing CodeT5 with data augmentation and post-processing of beams. We utilize GPT-3 with back translation for generation of synthetic examples. Further we apply rules of linear programming to score beams and correct beams based on common error patterns. We observe that with these enhancements CodeT5 base gives an execution accuracy of 0.73 which is significantly better than zero-shot execution accuracy of 0.41 by ChatGPT and 0.36 by Codex.

1 INTRODUCTION

Management of budget portfolio across various advertisement channels, procurement of raw materials and machinery in textile production, allocation of resources to a software or engineering project, distribution of vaccines and vaccination centers, etc., all these decision problems and many more across a variety of business and application domains can be modeled as optimization problems. Appropriate modeling can provide an immediate impact on profitability, safety, sustainability etc. In essence achieving maximum benefit from limited resources is the key for every successful business. However, typically a team of experts is required to identify the right decision variables, constraints and objectives in order to formulate the problem and solve it using a optimization solver. Therefore, only critical operations are given priority and optimized, leaving a large number of decisions unmodeled and therefore not optimized. Consequently automation in formulation of optimization problems can democratize the process and allow inclusion of broader set of decision variables and decision makers.

Recent years have seen significant advances in the use of neural networks for synthesis of both procedural e.g (Ahmad et al., 2021), (Wang et al., 2021) and (Devlin et al., 2017a) as well as declarative e.g. (Han et al., 2022) and (Zhong et al., 2017) programs. Optimization problems are similar to declarative programs but expressed with constraints and objectives. Just as done with declarative programs the execution is not to be determined by the developer; typically off the shelf solvers will be utilized and the developer needs to only specify the type of the solver to be used. Its possible to express declarative programs in high level languages like python, for example (Huang et al., 2021), (KIMMIG et al., 2011) are libraries used for logic programs, meanwhile (Mitchell et al., 2011), (Diamond & Boyd, 2016) are examples of libraries used for optimization problems.

Thus one who looks to autoformulate optimization problems may also desire automatic synthesis of code in a high level language using a library like PuLP or CVXPY. Nevertheless, despite these similarities there is a surprising dearth of work on automatic synthesis mathematical programs and the relevant code. The recent competition (Ramamonjison et al., 2022a) proposed at Neurips 22 is a notable exception.

In this paper, we envision a scenario where a business problem is communicated through natural language, and an autoformulation synthesizer must recognize the constraint and objectives. On account of a dearth of publicly available benchmark datasets we focus on the NL4OPT dataset which has defined tasks to help convert word problems to canonical form expression of linear programs. We show that by using proposed post processing and training methods we were able to achieve better canonical accuracy of 89.63 with a smaller model (CodeT5-base) compared to the larger (BART-large) model used by current SOTA 88.20 with lesser training epochs (30 vs 400).

The remainder of the paper is organized as follows: Section 2 summarizes the related work in literature. Section 3 covers the preliminaries, providing details about the NL4OPT dataset and the metrics of performance employed in this work. Section 4 describes the details of data augmentation and post processing used with CodeT5. Section 4 describes the instructions used with Codex and ChatGPT. Section 5 reports the primary experimental results and observations. Finally in Section 6 we summarize the key conclusions and discuss possibilities for future work.

2 RELATED WORK

Several recent works have utilized sequence to sequence models or pretrained Large Language Models for Code Generation tasks. (Devlin et al., 2017b), (Rossol, 1986) are examples of papers which train a model specific to synthetic data generated on the basis of a formal grammar. Its not always feasible to parse an entire corpus in terms of formal grammar and use that for tokenization. However, a number of LLMs like CodeBERT (Feng et al., 2020), CodeGPT (Lu et al., 2021), CodeT5 (Wang et al., 2021), are examples of models that have proven effective for python, C#, Java etc. code generation with just a BPE tokenization. These models are often fine-tuned for a downstream task. Meanwhile the emergence of significantly larger models like Codex, GPT-3 etc have popularized the notion of in context learning (Brown et al., 2020). In our work we explore both the modalities.

While many operations research related work has been seen over the years, very few focuses on auto-formulation of the optimization tasks from unstructured inputs such as text. Some of them addresses partial formulation tasks and solver configuration. Iommazzo et al. (2020) discuss methods to automatically configure solver parameters using Machine Learning. Kiziltan et al. (2016b) presents methods to extract constraints from natural language. In the paper MathoptInterface Legat et al. (2020) suggests a standard representation for optimization problems compatible with variety of solvers.

In a recent paper associated with the NL4OPT competitions, Ramamonjison et al. (2022a) have explained the use of transformers in extracting objective and constraints from natural language. We setup auto formulation as Mathematical program synthesis from natural language input. Code or program generated by LLMs can suffer from several lexical, syntactic and semantic errors. Therefore a number of papers use grammar guidance for decoding (Scholak et al. (2021), Jain et al. (2021)) or when suitable even use a symbolic or neural debugger for repairing the program (Ellis et al. (2019)).

Lack of data or parallel data is often a severe constraint in code generation and program synthesis tasks. Some of the approaches that have been adopted effectively include co-training,(Ahmad et al., 2021), that allows a model to learn a join representation space for NL as well as Code. Some models employ back translation, (Lachaux et al., 2020), to improve the pretraining or increase the data augmentation for training. Recent works (Sahu et al., 2022) and (Yoo et al., 2021) have also explored the use of LLMs for generation of synthetic data which can be used for data augmentation.

3 PRELIMINARIES

3.1 NL4OPT

We use NL4OPT (Ramamonjison et al., 2022a) dataset along with generated data to train our model and use only NL4OPT for evaluation. Data released as part of the NL4OPT competition consists three sets of word problems for linear programming from six different domains [Investment, Advertisement, Sales, Health Science, Transportation, Production]split into train(713), dev(99), and a private dataset test(290). Each linear word problem is accompanied by entity mapping and variable ordering. The labels are the Intermediate representation of the problem in either JSON format or XML which contains a list of constraints and a linear objective.

Listing 1: Example sales optimization problem from the dataset

```
A sports warehouse stocks rafts and kayaks. Each raft takes
10 sq ft of space while each kayak takes 12 sq ft of space.
The warehouse has 400 sq ft of space available. The warehouse
has a budget of $10000 with each raft costing $200 and each
kayak costing $250. With rafting being much more popular in
the area, at least 55% of all items in stock must be rafts.
If the profit per raft is $45 and the profit per kayak
is $55, how many of each should be bought and sold
to maximize profit?
```

Let $\mathrm{P}_{\text{train}}$, P_{dev} be the word problems in the train set and dev set respectively. Each problem in these sets has an objectives and 2 to 4 constraints . Please refer to the table 1 below for different constraint types.

$$\forall Obj : Obj_{\text{type}} \in [linear, sum]$$

$$\forall Con : Con_{\text{type}} \in (linear, sum, ratio, xy, xby, upperbound, lowerbound)$$

Constraint Type	**Mathematical representation**
sum	x + y $\leq c$
upperbound	x $\leq c$
lowerbound	x $\geq c$
linear	a_1 x + a_2 y $\leq c$
ratio	x $\leq c(x + y)$
xby	x $\leq ay$
xy	x $\leq y$

Table 1: Constraint Types used

3.2 EVALUATION METRICS

3.2.1 CANONICAL ACCURACY - AS IN NL4OPT EVALUATION

The evaluation is based on the correct declaration of objectives and constraints. Accuracy is calculated using the following formula:

$$CanonicalAccuracy = 1 - \frac{\sum(FP + FN)}{\sum D} \tag{1}$$

where FP denotes the number of predicted objectives or constraints not matching with any of the actual declarations, FN denotes the number of actual declarations not matching with any of the predicted declarations and D denoted the total number of declarations in the ground truth.

3.2.2 EXECUTION ACCURACY

The above canonical metrics expect the language model to adhere to the same order of variable mappings as the labels. Meaning $a.X + b.Y \leq c$ is considered different from $b.Y + a.X \leq c$. To ensure the variable order is maintained an ordered variable mapping is expected as an input.

The fine-tuned models were able to adhere to the variable ordering if given as input, however, Codex, and ChatGPT were not able to adhere to the same and hallucinated variable names often. This made comparing canonical accuracy very difficult. Hence we propose using execution accuracy to compare fine tuned and GPT models.

To measure execution accuracy, we convert both actual and predicted declarations to python programs which solve for optimal values using Pulp (Mitchell et al., 2011). We then compare the optimal values for exact match. This metric is used to compare program-only outputs such as codex.

4 METHODOLOGIES FOR FINE-TUNED MODELS

We experimented with a variety of transformer based language models such as CodeT5, Bloom, Flan-T5 of various sizes. The common setup is to generate the intermediate representation in JSON format with the word problem as the input. One exception is OpenAI Codex (Chen et al., 2021), which generated a python program using pulp library to formulate the problem. The performance of fine-tuned models were measured using the canonical accuracy of the IR generated and using execution match in the case of Codex.

4.1 FINE TUNING FOR TARGET AND AUXILLARY TASKS

The models are trained for the target task which is to predict the intermediate representation of the problem in JSON format which then can be used to fill the program sketch created for linear optimization problems. We observed that out of the models we selected, CodeT5 (Wang et al., 2021) trained on code generation tasks was performing better than others in our preliminary training (trained with only NL4OPT training data for 20 epochs). Hence we decided to take CodeT5 variants (small, base, large) as our base models for our experiments. Refer Appendix Table:6

It has been proven that models trained on related subtasks such as scratchpads (Nye et al., 2021), and Chain of thought (Wei et al., 2022), improve on the target tasks as well. We trained our models on the primary task which is to generate the declarations [objectives, constraints] as well as on subtasks such as (i) Predicting the number of constraints in the problem, (ii) Predicting the variable names,(iii) Predicting the parameter values,(iv) Direction of the objective.

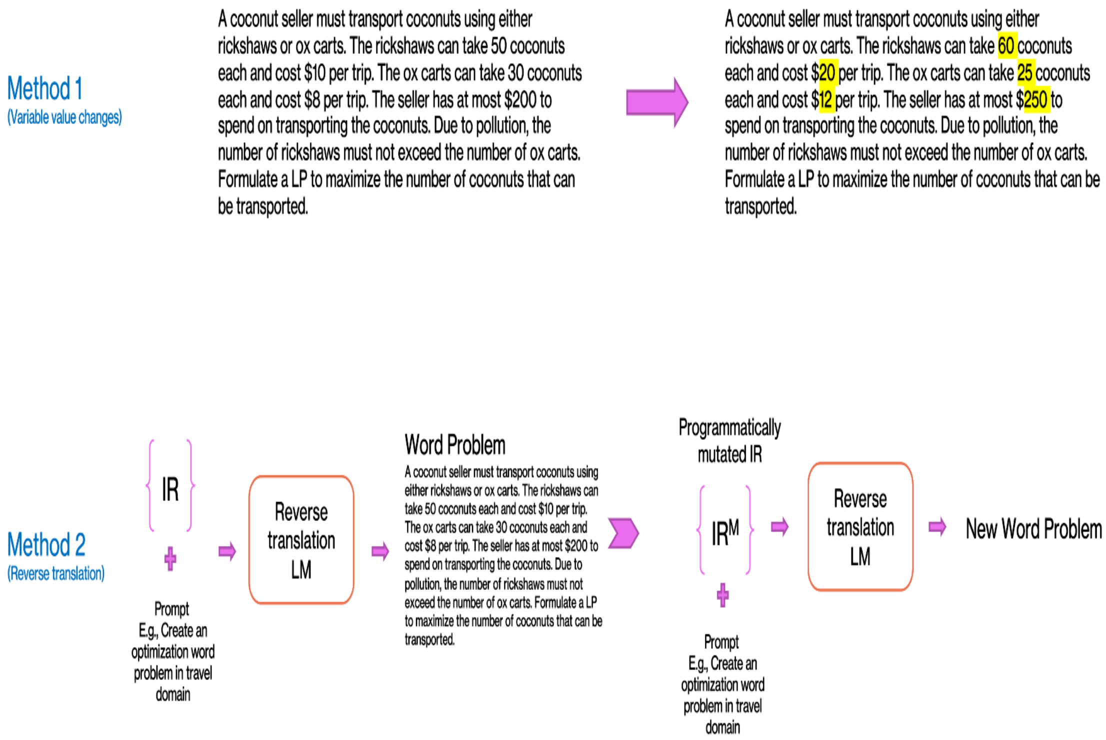
All models were trained for 30 epochs on A100 40GB, with a learning rate of 5e-5, effective batch size of 32, and weight decay of 1e-5 with cross-entropy loss. The nl4opt data is provided with a entity tagging (output of a fine-tuned NER Model) used to enrich the text input. We train the models with both enriched and original data to minimize the dependency on the NER models.

4.2 AUGMENTATION USING LANGUAGE MODELS

The Natural Language for Optimization Problems (NL4OPT) dataset was the only available resource that featured optimization problems described in natural language. However, the dataset had several limitations, including the fact that it only contained single linear objective problems and was limited to a few (6) industrial domains. Given that many real-world business problems are far more complex and involve multiple objectives, we decided to augment the dataset through parameter mutation and data synthesis utilizing the Generative Pre-trained Transformer 3 (GPT3) models. We observed that the generated word problems were not semantically correct, however, it introduced variations in text and domains so that the trained model performed better on the validation dataset. Refer Figure1

The utilization of back translation through large language models, instead of templates, was motivated by the desire to simulate problems from various domains and to incorporate variations in the input text. The process involved transforming the intermediate representations from the NL4OPT dataset by mutating variables and parameter. The mutated intermediate representations were then utilized to generate human-readable problem descriptions through GPT3, which were validated and subsequently added to the training set. To imitate multi-objective problems, intermediate representations of complex single-objective problems with more than three linear constraints were selected and one of the linear constraints was converted into an additional objective simply by removing the limit.

The final augmented dataset used for training included both single-objective and multi-objective problems, whereas, the test data is originally from NL4OPT dataset which only consisted of single objective problem.



A coconut seller must transport coconuts using either rickshaws or ox carts. The rickshaws can take 50 coconuts each and cost $10 per trip. The ox carts can take 30 coconuts each and cost $8 per trip. The seller has at most $200 to spend on transporting the coconuts. Due to pollution, the number of rickshaws must not exceed the number of ox carts. Formulate a LP to maximize the number of coconuts that can be transported.

A coconut seller must transport coconuts using either rickshaws or ox carts. The rickshaws can take 60 coconuts each and cost $20 per trip. The ox carts can take 25 coconuts each and cost $12 per trip. The seller has at most $250 to spend on transporting the coconuts. Due to pollution, the number of rickshaws must not exceed the number of ox carts. Formulate a LP to maximize the number of coconuts that can be transported.

A coconut seller must transport coconuts using either rickshaws or ox carts. The rickshaws can take 50 coconuts each and cost $10 per trip. The ox carts can take 30 coconuts each and cost $8 per trip. The seller has at most $200 to spend on transporting the coconuts. Due to pollution, the number of rickshaws must not exceed the number of ox carts. Formulate a LP to maximize the number of coconuts that can be transported.

Figure 1: Data Augmentation methods

	Original	Simulated	Total
Single Objective	713	3553	4226
Multi Objective	0	483	483
Total			4749

Table 2: Augmented Training Data

4.3 POST INFERENCE PROCESSING

Optimization is prevalent across various industries and encompasses a diverse range of domain rules and knowledge, much of which may not be explicitly stated in the problem description but is nonetheless expected to be implicitly followed by the system. To address this, we employ symbolic post-processing to accurately select the appropriate sample from the generated outputs and to correct any known errors in the selected candidate. This approach enhances the reliability and robustness of the generated solutions. See in Appendix (figure 3)

4.3.1 CUSTOM BEAM SCORER

Recent studies by Hokamp & Liu (2017) shows that constrained beam search improves accuracy of the generated text, We generate multiple samples using the models and along with the probability score from the model, used in combination execution guided scoring and rule based scoring. We observe accuracy improvements by applying few general semantics of Linear programming expressed as rules. These rules can be easily extended to include domain specific rules from knowledge bases or from domain text similar to the work by Kiziltan et al. (2016b)

To give an example of these rules, Let constraint Con(W,x,z) where W is the coefficient matrix, x is the variable and z is the RHS of the inequality. Then the based on simple linear equation properties, we know that:

$$\forall Con(W, x, z) : con1 \neq con2 \land w1 \geq w2 \rightarrow z1 \geq z2$$
$$\forall Con(W, x, z) : con1 \neq con2 \rightarrow w1 \neq w2$$

For each such rule violated, a penalty is applied on the corresponding beam, reducing the chance of that beam being selected. We then select 5 top candidates and calculate canonical accuracy to select the best one.

4.3.2 AUTOMATIC POST EDITING USING DOMAIN RULES

We perform certain corrections based on the domain rules, for example, if the objective function has the same coefficient for all the decision variables, it could be considered as a sum function rather than as linear.

$$\forall Obj(W, x) : w_1 == w_2 - > Obj_{t}ype =' objvar'$$

It was also beneficial to apply sanitizing such as duplicate constraint removal, removing terms when the constraint type is ”sum” etc. Again, this rule set can be extended to have domain specific corrections and can operate completely decoupled from the model itself. This enables scaling of the model application to much wider industries.

4.4 CONFIGURING OPTIMIZATION SOLVERS

We created an adapter script to load the intermediate format (from the model) into an Optimization solver. We chose PULP as our optimization library since it had simple interfaces to configure the LP problem.

Solving for the optimized values enables newer evaluation/beam selection methods. The optimal values from LLMs + Adapter can be compared with heuristics created by domain experts or generated from historical data. Using this method also enabled us to compare results with Codex GPT3 outputs since in Codex outputs, decision variable order was not strictly followed by the generated programs which made Canonical comparison difficult.

4.5 ZERO SHOT SYNTHESIS USING CODEX AND CHATGPT

The intermediate representations were difficult for the LLMs such as Codex and ChatGPT even when prompted with semantically selected few shot examples. Hence, instead of prompting to generate the intermediate representation, we prompted Codex and ChatGPT to generate programs (refer 2) that can solve the given optimization problem. The models tend to use a variety of optimization solvers (CVXPY, Pulp, Scipy). To maintain uniformity, We prompted to use Pulp as the default solver and a default variable name ”prob” to hold the problem formulation (refer listing 2).

ChatGPT (`https://chat.openai.com/`) being a conversational AI, generated additional explanations which were ignored and only generated code was extracted. Refer Appendix figure: 4 and 5.

Using the adapter discussed in 4.4 the actual declarations can be converted to similar python programs. The generated programs (from actual and from LLMs) when executed results in optimal values for the problem objective. The results of the predicted and actual programs are then compared for execution matches.

Listing 2: Example of program generated by Codex. Prompt text is enclosed in PROMPT tag for easy understanding

```
#<PROMPT>
”””
 A sports warehouse stocks rafts and kayaks. Each raft takes
 10 sq ft of space while each kayak takes 12 sq ft of space.
 The warehouse has 400 sq ft of space available. The warehouse
 has a budget of $10000 with each raft costing $200 and each
 kayak costing $250. With rafting being much more popular in
 the area, at least 55% of all items in stock must be rafts.
 If the profit per raft is $45 and the profit per kayak
 is $55, how many of each should be bought and sold
 to maximize profit?
”””
# Use Pulp to solve the problem ”prob_−1757358180.lp”
# Use the provided variable names rafts,kayaks
from pulp import ∗
# Create the ”prob” variable to contain the problem data
problem_name = ”prob_−1757358180”
prob = LpProblem(problem_name,
#<PROMPT>
                     LpMaximize)
# The 2 variables rafts and kayaks are created with a
# lower limit of 0
rafts = LpVariable(”rafts”,
                      lowBound=0,
                      cat=’Integer’) # @UndefinedVariable
kayaks = LpVariable(”kayaks”,
                       lowBound=0,
                       cat=’Integer’) # @UndefinedVariable
# The objective function is added to ”prob” first
prob += 45 ∗ rafts + 55 ∗ kayaks, ”Profit”
# The two constraints are entered
prob += 10 ∗ rafts + 12 ∗ kayaks <= 400, ”Space”
prob += 200 ∗ rafts + 250 ∗ kayaks <= 10000, ”Cost”
prob += 0.55 ∗ (rafts + kayaks) <= rafts, ”Percentage”
# The problem data is written to an .lp file
prob.writeLP(problem_name + ”.lp”)
# The problem is solved using PuLP’s choice of Solver
prob.solve() # @UndefinedVariable
# The status of the solution is printed to the screen
print(”Status:”, LpStatus[prob.status]) # @UndefinedVariable
# Each of the variables is printed with
# it’s resolved optimum value
for v in prob.variables(): # @UndefinedVariable
    print(v.name, ”=”, v.varValue)
```

5 RESULTS

We observed that codeT5-base model with custom beam scoring and correction performed better than other fine tuned models.

To validate the new problems generated using back translation, we ensured that the parameters, decision variables, and objective/constraint directions were accurately represented in the text. However, we discovered that the word problems generated were not semantically correct in most cases. Hence, even though the models were trained on the augmented data, they were only evaluated using the original NL4OPT dataset. Our findings showed that training the models with the augmented data even when they are noisy, led to improved performance on the actual test data.

Ablation study shows that Subtask augmentation boosts the accuracy up by 10 percent in certain models. Similarly, training with noisy back translated data also boosts the accuracy by 9 points. Our rule based correction and scoring mechanism helps smaller models to achieve accuracy comparable with SOTA. We also show that with pass@k methods, we were able to surpass SOTA.

	code-t5-small	**code-t5-base**	**code-t5-large**
No Augmentation	0.47	0.57	0.62
Sub task Augmentation	0.6	0.69	0.72
Back translation	0.69	0.78	0.78
Sub Task + Back Translation	0.73	0.81	0.82
Correction + Beam Search	0.7435	0.85	0.87
Correction + Beam search (custom scoring)	0.74	0.8812	0.87

Table 3: Ablation Study on codeT5: Figure shows the accuracy levels reached by various models with a beam size of 5. We can observe that custom beam search helps the base model reach the accuracy similar to the larger ones.

The state of the art model (Gangwar & Kani, 2022) achieved accuracy of 0.882 from 5 beams on the dev set after 400 training epochs using BART large models without any data augmentation. We were able to achieve accuracy of 0.893 with pass@k (as defined in Chen et al. (2021)) where k=5 beams with 30 training epochs by using data augmentation and post model symbolic methods. We see 3% to 5% increase in accuracy during inference time when logic guided corrections and custom beam search are used together. Further the percentage gain from the symbolic knowledge is more for larger beam sizes (refer Figure 3.), however after beam size 8, the accuracy started reducing.

Model Name	**Size**	**Epochs**	**pass@k**	**Canonical Acc.**	**Method**
Baseline	140M	200	-	63	Prompt Guided
Current SOTA	400M	400	5	88.2	All at once
Proposed Model	220M	30	5	**89.63**	Logic guided Correction

Table 4: Comparison of Baseline (BART), Current SOTA (Bart Large) Gangwar & Kani (2022), Proposed (CodeT5 + Logic Guided Correction) performances

We also compared the best model with Codex and ChatGPT outputs on exact match of optimal values since there is no order mapping of variables possible for codex outputs which makes canonical comparison difficult. We can see that the fine tuned model with logic guided scoring and correction was able to outperform Codex. We can infer that learning the intermediate representation enables models to more accurately solve the problem. Codex was able to score 36% on exact match, ChatGPT was able to score 41%, whereas our finetuned models were able to achieve 73% exact match.

	codeT5	**Codex**	**ChatGPT**
Execution Match	0.73	0.36	0.41

Table 5: Execution match is calculated by executing the generated program to find the optimal objective value and finding exact match against actual

6 CONCLUSION

In this paper, we considered the task of formulating optimization problems from natural language. The dataset we considered had domain-specific variations, and thus with only 730 training samples, the data can be considered to be sparse. We evaluated several models and proposed methods that enable the use of large language models to synthesize the canonical forms as well as python code despite the task being a low-data task. We observed progressive utility for the use of auxiliary tasks, GPT-3 based data augmentation, and logic based post-processing of decoding beams. Since we extended the task to code synthesis, as against just classification as done in the NL4OPT challenge, we were able to measure the efficacy of the methods in terms of execution accuracy.

REFERENCES

Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation, 2021. URL `https://arxiv.org/abs/2103.06333`.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, 2007.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

Hadrien Cambazard, Nicolas Catusse, Nadia Brauner, and Pierre Lemaire. Teaching OR: automatic evaluation for linear programming modelling. *4OR*, 20(2):333–345, July 2021. doi: 10.1007/s10288-021-00488-9. URL `https://doi.org/10.1007/s10288-021-00488-9`.

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. URL `https://arxiv.org/abs/2107.03374`.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022. URL `https://arxiv.org/abs/2210.11416`.

Jacob Devlin, Rudy Bunel, Rishabh Singh, Matthew Hausknecht, and Pushmeet Kohli. Neural program meta-induction, 2017a. URL `https://arxiv.org/abs/1710.04157`.

Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o, 2017b. URL `https://arxiv.org/abs/1703.07469`.

Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization, 2016. URL `https://arxiv.org/abs/1603.00943`.

Alexander Elizarov, Alexander Kirillovich, Evgeny Lipachev, Olga Nevzorova, Valery Solovyev, and Nikita Zhiltsov. Mathematical knowledge representation: Semantic models and formalisms, 2014. URL `https://arxiv.org/abs/1408.6806`.

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl, 2019. URL `https://arxiv.org/abs/1906.04604`.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages, 2020. URL `https://arxiv.org/abs/2002.08155`.

Neeraj Gangwar and Nickvash Kani. Highlighting named entities in input for auto-formulation of optimization problems, 2022. URL `https://arxiv.org/abs/2212.13201`.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, David Peng, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Shafiq Joty, Alexander R. Fabbri, Wojciech Kryscinski, Xi Victoria Lin, Caiming Xiong, and Dragomir Radev. Folio: Natural language reasoning with first-order logic, 2022. URL `https://arxiv.org/abs/2209.00840`.

Pascal Van Hentenryck. The opl optimization programming language. 1999.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1141. URL `https://aclanthology.org/P17-1141`.

Sitao Huang, Kun Wu, Hyunmin Jeong, Chengyue Wang, Deming Chen, and Wen-Mei Hwu. Pylog: An algorithm-centric python-based fpga programming and synthesis flow. *IEEE Transactions on Computers*, 70(12):2015–2028, 2021. doi: 10.1109/TC.2021.3123465.

Gabriele Iommazzo, Claudia D'Ambrosio, Antonio Frangioni, and Leo Liberti. A learning-based mathematical programming formulation for the automatic configuration of optimization solvers. In *Machine Learning, Optimization, and Data Science*, pp. 700–712. Springer International Publishing, 2020. doi: 10.1007/978-3-030-64583-0_61. URL `https://doi.org/10.1007/978-3-030-64583-0_61`.

Naman Jain, Skanda Vaidyanath, Arun Iyer, Nagarajan Natarajan, Suresh Parthasarathy, Sriram Rajamani, and Rahul Sharma. Jigsaw: Large language models meet program synthesis, 2021. URL `https://arxiv.org/abs/2112.02969`.

ANGELIKA KIMMIG, BART DEMOEN, LUC DE RAEDT, VÍ TOR SANTOS COSTA, and RICARDO ROCHA. On the implementation of the probabilistic logic programming language ProbLog. *Theory and Practice of Logic Programming*, 11(2-3):235–262, jan 2011. doi: 10.1017/s1471068410000566. URL `https://doi.org/10.1017%2Fs1471068410000566`.

Zeynep Kiziltan, Marco Lippi, and Paolo Torroni. Constraint detection in natural language problem descriptions. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pp. 744–750. AAAI Press, 2016a. ISBN 9781577357704.

Zeynep Kiziltan, Marco Lippi, and Paolo Torroni. Constraint detection in natural language problem descriptions. *Neural Computation*, 18:1527–1554, 2016b.

Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages, 2020. URL `https://arxiv.org/abs/2006.03511`.

Benoit Legat, Oscar Dowson, Joaquim Dias Garcia, and Miles Lubin. Mathoptinterface: a data structure for mathematical optimization problems, 2020. URL `https://arxiv.org/abs/2002.03447`.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation, 2021. URL `https://arxiv.org/abs/2102.04664`.

Stuart Mitchell, Michael J. O'Sullivan, and Iain Dunning. Pulp : A linear programming toolkit for python. 2011.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021. URL `https://arxiv.org/abs/2112.00114`.

Bahadorreza Ofoghi, Vicky Mak, and John Yearwood. A knowledge representation approach to automated mathematical modelling, 2020. URL `https://arxiv.org/abs/2011.06300`.

Axel Parmentier. Learning structured approximations of combinatorial optimization problems, 2021. URL `https://arxiv.org/abs/2107.04323`.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models, 2022. URL `https://arxiv.org/abs/2201.11227`.

Rindranirina Ramamonjison, Haley Li, Timothy T. Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions, 2022a. URL `https://arxiv.org/abs/2209.15565`.

Rindranirina Ramamonjison, Haley Li, Timothy T. Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions, 2022b. URL `https://arxiv.org/abs/2209.15565`.

Yara Rizk, Praveen Venkateswaran, Vatche Isahagian, and Vinod Muthusamy. A case for business process-specific foundation models, 2022. URL `https://arxiv.org/abs/2210.14739`.

L. Rossol. The KAREL language for programmable automation. In *Initiativen für die Fabrik mit Zukunft*, pp. 293–312. Springer Berlin Heidelberg, 1986. doi: 10.1007/978-3-662-12068-2_18. URL `https://doi.org/10.1007/978-3-662-12068-2_18`.

Gaurav Sahu, Pau Rodriguez, Issam H. Laradji, Parmida Atighehchian, David Vazquez, and Dzmitry Bahdanau. Data augmentation for intent classification with off-the-shelf large language models, 2022. URL `https://arxiv.org/abs/2204.01959`.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models, 2021. URL `https://arxiv.org/abs/2109.05093`.

Pascal Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0262720302.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation, 2021. URL `https://arxiv.org/abs/2109.00859`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2022. URL `https://arxiv.org/abs/2201.11903`.

BigScience Workshop, :, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, Dragomir Radev, Eduardo González Ponferrada, Efrat Levkovizh, Ethan Kim, Eyal Bar Natan, Francesco De Toni, Gérard Dupont, Germán Kruszewski, Giada Pistilli, Hady Elsahar, Hamza Benyamina, Hieu Tran, Ian Yu, Idris Abdulmumin, Isaac Johnson, Itziar Gonzalez-Dios, Javier de la Rosa, Jenny Chim, Jesse Dodge, Jian Zhu, Jonathan Chang, Jörg Frohberg, Joseph Tobing, Joydeep Bhattacharjee, Khalid Almubarak, Kimbo Chen, Kyle Lo, Leandro Von Werra, Leon Weber, Long Phan, Loubna Ben allal, Ludovic Tanguy, Manan Dey, Manuel Romero Muñoz, Maraim Masoud, María Grandury, Mario Šaško, Max Huang, Maximin Coavoux, Mayank Singh, Mike Tian-Jian Jiang, Minh Chien Vu, Mohammad A. Jauhar, Mustafa Ghaleb, Nishant Subramani, Nora Kassner, Nurulaqilla Khamis, Olivier Nguyen, Omar Espejel, Ona de Gibert, Paulo Villegas, Peter Henderson, Pierre Colombo, Priscilla Amuok, Quentin Lhoest, Rheza Harliman, Rishi Bommasani, Roberto Luis López, Rui Ribeiro, Salomey Osei, Sampo Pyysalo, Sebastian Nagel, Shamik Bose, Shamsuddeen Hassan Muhammad, Shanya Sharma, Shayne Longpre, Somaieh Nikpoor, Stanislav Silberberg, Suhas Pai, Sydney Zink, Tiago Timponi Torrent, Timo Schick, Tristan Thrush, Valentin Danchev, Vassilina Nikoulina, Veronika Laippala, Violette Lepercq, Vrinda Prabhu, Zaid Alyafeai, Zeerak Talat, Arun Raja, Benjamin Heinzerling, Chenglei Si, Davut Emre Taşar, Elizabeth Salesky, Sabrina J. Mielke, Wilson Y. Lee, Abheesht Sharma, Andrea Santilli, Antoine Chaffin, Arnaud Stiegler, Debajyoti Datta, Eliza Szczechla, Gunjan Chhablani, Han Wang, Harshit Pandey, Hendrik Strobelt, Jason Alan Fries, Jos Rozen, Leo Gao, Lintang Sutawika, M Saiful Bari, Maged S. Al-shaibani, Matteo Manica, Nihal Nayak, Ryan Teehan, Samuel Albanie, Sheng Shen, Srulik Ben-David, Stephen H. Bach, Taewoon Kim, Tali Bers, Thibault Fevry, Trishala Neeraj, Urmish Thakker, Vikas Raunak, Xiangru Tang, Zheng-Xin Yong, Zhiqing Sun, Shaked Brody, Yallow Uri, Hadar Tojarieh, Adam Roberts, Hyung Won Chung, Jaesung Tae, Jason Phang, Ofir Press, Conglong Li, Deepak Narayanan, Hatim Bourfoune, Jared Casper, Jeff Rasley, Max Ryabinin, Mayank Mishra, Minjia Zhang, Mohammad Shoeybi, Myriam Peyrounette, Nicolas Patry, Nouamane Tazi, Omar Sanseviero, Patrick von Platen, Pierre Cornette, Pierre François Lavallée, Rémi Lacroix, Samyam Rajbhandari, Sanchit Gandhi, Shaden

Smith, Stéphane Requena, Suraj Patil, Tim Dettmers, Ahmed Baruwa, Amanpreet Singh, Anastasia Cheveleva, Anne-Laure Ligozat, Arjun Subramonian, Aurélie Névéol, Charles Lovering, Dan Garrette, Deepak Tunuguntla, Ehud Reiter, Ekaterina Taktasheva, Ekaterina Voloshina, Eli Bogdanov, Genta Indra Winata, Hailey Schoelkopf, Jan-Christoph Kalo, Jekaterina Novikova, Jessica Zosa Forde, Jordan Clive, Jungo Kasai, Ken Kawamura, Liam Hazan, Marine Carpuat, Miruna Clinciu, Najoung Kim, Newton Cheng, Oleg Serikov, Omer Antverg, Oskar van der Wal, Rui Zhang, Ruochen Zhang, Sebastian Gehrmann, Shachar Mirkin, Shani Pais, Tatiana Shavrina, Thomas Scialom, Tian Yun, Tomasz Limisiewicz, Verena Rieser, Vitaly Protasov, Vladislav Mikhailov, Yada Pruksachatkun, Yonatan Belinkov, Zachary Bamberger, Zdeněk Kasner, Alice Rueda, Amanda Pestana, Amir Feizpour, Ammar Khan, Amy Faranak, Ana Santos, Anthony Hevia, Antigona Unldreaj, Arash Aghagol, Arezoo Abdollahi, Aycha Tammour, Azadeh HajiHosseini, Bahareh Behroozi, Benjamin Ajibade, Bharat Saxena, Carlos Muñoz Ferrandis, Danish Contractor, David Lansky, Davis David, Douwe Kiela, Duong A. Nguyen, Edward Tan, Emi Baylor, Ezinwanne Ozoani, Fatima Mirza, Frankline Ononiwu, Habib Rezanejad, Hessie Jones, Indrani Bhattacharya, Irene Solaiman, Irina Sedenko, Isar Nejadgholi, Jesse Passmore, Josh Seltzer, Julio Bonis Sanz, Livia Dutra, Mairon Samagaio, Maraim Elbadri, Margot Mieskes, Marissa Gerchick, Martha Akinlolu, Michael McKenna, Mike Qiu, Muhammed Ghauri, Mykola Burynok, Nafis Abrar, Nazneen Rajani, Nour Elkott, Nour Fahmy, Olanrewaju Samuel, Ran An, Rasmus Kromann, Ryan Hao, Samira Alizadeh, Sarmad Shubber, Silas Wang, Sourav Roy, Sylvain Viguier, Thanh Le, Tobi Oyebade, Trieu Le, Yoyo Yang, Zach Nguyen, Abhinav Ramesh Kashyap, Alfredo Palasciano, Alison Callahan, Anima Shukla, Antonio Miranda-Escalada, Ayush Singh, Benjamin Beilharz, Bo Wang, Caio Brito, Chenxi Zhou, Chirag Jain, Chuxin Xu, Clémentine Fourrier, Daniel León Periñán, Daniel Molano, Dian Yu, Enrique Manjavacas, Fabio Barth, Florian Fuhrimann, Gabriel Altay, Giyaseddin Bayrak, Gully Burns, Helena U. Vrabec, Imane Bello, Ishani Dash, Jihyun Kang, John Giorgi, Jonas Golde, Jose David Posada, Karthik Rangasai Sivaraman, Lokesh Bulchandani, Lu Liu, Luisa Shinzato, Madeleine Hahn de Bykhovetz, Maiko Takeuchi, Marc Pàmies, Maria A Castillo, Marianna Nezhurina, Mario Sänger, Matthias Samwald, Michael Cullan, Michael Weinberg, Michiel De Wolf, Mina Mihaljcic, Minna Liu, Moritz Freidank, Myungsun Kang, Natasha Seelam, Nathan Dahlberg, Nicholas Michio Broad, Nikolaus Muellner, Pascale Fung, Patrick Haller, Ramya Chandrasekhar, Renata Eisenberg, Robert Martin, Rodrigo Canalli, Rosaline Su, Ruisi Su, Samuel Cahyawijaya, Samuele Garda, Shlok S Deshmukh, Shubhanshu Mishra, Sid Kiblawi, Simon Ott, Sinee Sang-aroonsiri, Srishti Kumar, Stefan Schweter, Sushil Bharati, Tanmay Laud, Théo Gigant, Tomoya Kainuma, Wojciech Kusa, Yanis Labrak, Yash Shailesh Bajaj, Yash Venkatraman, Yifan Xu, Yingxin Xu, Yu Xu, Zhe Tan, Zhongli Xie, Zifan Ye, Mathilde Bras, Younes Belkada, and Thomas Wolf. Bloom: A 176b-parameter open-access multilingual language model, 2022. URL `https://arxiv.org/abs/2211.05100`.

Kang Min Yoo, Dongju Park, Jaewook Kang, Sang-Woo Lee, and Woomyeong Park. Gpt3mix: Leveraging large-scale language models for text augmentation, 2021. URL `https://arxiv.org/abs/2104.08826`.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017. URL `https://arxiv.org/abs/1709.00103`.

7 APPENDIX

7.1 INITIAL MODEL SELECTION

We selected CodeT5 for our experiment since we setup our problem as a program synthesis task and it performed better compared to other similar sized models.

Model Name	Inputs	Output	Canonical accuracy	Rouge	Epochs
code-t5-small	Doc	JSON	47%	83%	30
code-t5-small	Doc + Vars + Ord	JSON	52%	82%	30
code-t5-base	Doc	JSON	57%	83%	20
code-t5-base	Doc + Vars + Ord	JSON	64%	84%	20
code-t5-large	Doc	JSON	67%	NA	10
flan-t5-large	Doc	JSON	27%	79%	10
gpt3-curie	Doc	JSON	17%	NA	4

Table 6: All models trained on unaugmented data, small models for 30 epochs, base ofr 20 and large ones for 10 epochs. The type of input specifies what texts were concatenated from the input records.

7.2 BEAM SIZE VS CANONICAL ACCURACY

We see that with logic guided corrections, larger beam sizes until a certain size gain accuracy compared to its non guided pair.

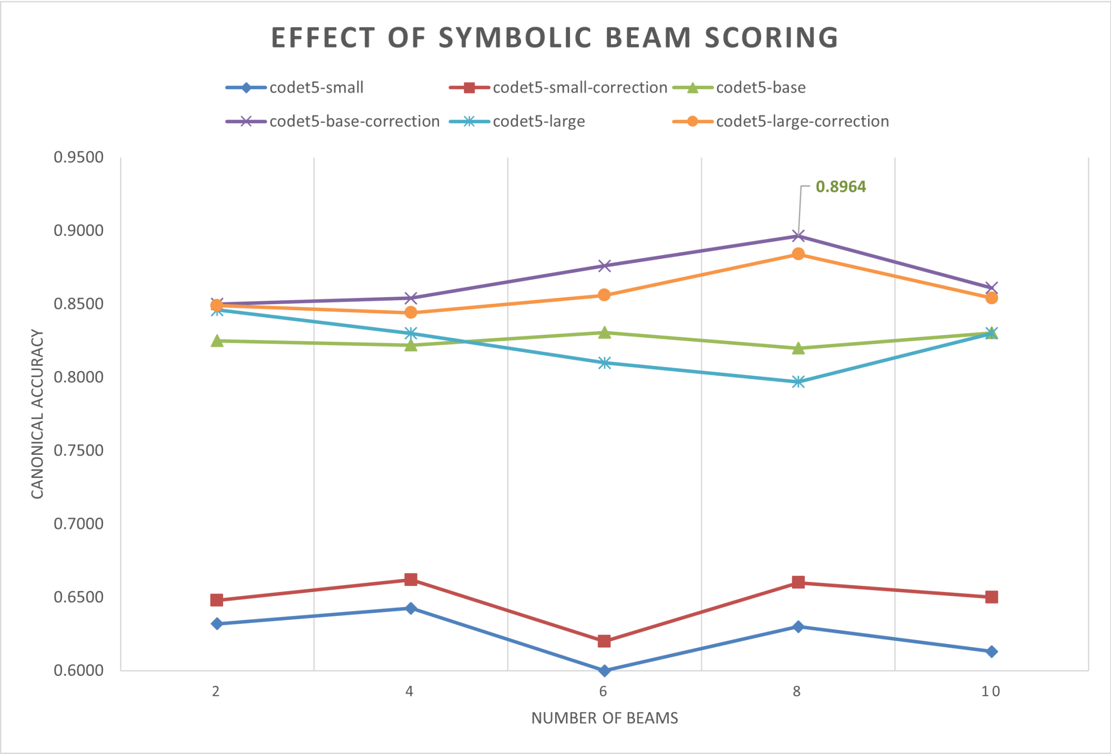


Figure 2: Effect of custom beam search on Accuracy

7.3 ERROR ANALYSIS

We will discuss frequently seen errors in this section, we believe that most of them can be addressed with carefully designed rules and data augmentation.

Missing variables in the objective.

```
Actual : {’obj_declaration’: {’type’: ’objective’, ’direction’: ’
    maximize’, ’name’: ’viewers’, ’terms’: {’magazine␣ad’: ’10000’
    , ’flyer’: ’5000’, ’billboard␣ad’: ’25000’}}
Predicted: {’obj_declaration’: {’type’: ’objective’, ’direction’:
    ’maximize’, ’name’: ’viewers’, ’terms’: {’flyer’: ’5000’, ’
    billboard␣ad’: ’25000’}}
```

Correct Prediction, however label is missing a constraint: As we could see the constraint that a train could atmost carry 500 passangers is not in the original label. Similarly we found 2 more cases where predicted actually was correct.

```
text:’A train can carry at most 500 passengers. It has two seat
    types: AC seats, and non-AC seats (AC is air conditioned). A
    profit of $50 is made on each AC seat ticket and a profit of
    $30 is made on each non-AC seat ticket. The train company
    reserves at least 100 seats as AC seats. However, a minimum of
     2 times as many passengers prefer to travel on non-AC seats
    than on AC seats. How many seat tickets of each type should be
     sold to maximize profit? What is that profit?’

Actual:
{’obj_declaration’: {’type’: ’objective’, ’direction’: ’maximize’,
     ’name’: ’profit’, ’terms’: {’AC seat’: ’50’, ’non-AC seat’:
    ’30’}}, ’const_declarations’: [{’type’: ’lowerbound’, ’
    direction’: ’at least’, ’limit’: ’100’, ’var’: ’AC seats’, ’
    operator’: ’GREATER_OR_EQUAL’}, {’type’: ’xby’, ’x_var’: ’non-
    AC seats’, ’direction’: ’minimum’, ’param’: ’2’, ’y_var’: ’AC
    seats’, ’operator’: ’GREATER_OR_EQUAL’}], ’vars’: [’AC seats’,
     ’non-AC seats’]}
---------------------------------------
Predicted
{’obj_declaration’: {’type’: ’objective’, ’direction’: ’maximize’,
     ’name’: ’profit’, ’terms’: {’AC seat’: ’50’, ’non-AC seat’:
    ’30’}}, ’const_declarations’: [{’type’: ’sum’, ’direction’: ’
    at most’, ’limit’: ’500’, ’operator’: ’LESS_OR_EQUAL’}, {’type
    ’: ’lowerbound’, ’direction’: ’at least’, ’limit’: ’100’, ’var
    ’: ’AC seats’, ’operator’: ’GREATER_OR_EQUAL’}, {’type’: ’xby
    ’, ’x_var’: ’non-AC seats’, ’direction’: ’minimum’, ’param’:
    ’2’, ’y_var’: ’AC seats’, ’operator’: ’GREATER_OR_EQUAL’}], ’
    vars’: [’AC seats’, ’non-AC seats’], ’id’: ’-996226930’}
```

Missing Constraint:In few cases, the predicted missed generating a constraint all together in all the beams generated.

```
Actual:
{’obj_declaration’: {’type’: ’objvar’, ’direction’: ’minimize’, ’
    name’: ’total number of stores’, ’vars’: [’film-based’, ’
    electrical-based stores’]}, ’const_declarations’: [{’type’: ’
    xby’, ’x_var’: ’electrical-based stores’, ’direction’: ’at
    least’, ’param’: ’two’, ’y_var’: ’film-based stores’, ’
    operator’: ’GREATER_OR_EQUAL’}, {’type’: ’lowerbound’, ’
    direction’: ’at least’, ’limit’: ’5’, ’var’: ’film-based
    stores’, ’operator’: ’GREATER_OR_EQUAL’}, {’type’: ’linear’, ’
    direction’: ’at least’, ’limit’: ’170’, ’terms’: {’Film-based
    stores’: ’2’, ’electrical-based store’: ’four’}, ’operator’: ’
    GREATER_OR_EQUAL’}, {’type’: ’linear’, ’direction’: ’at most’,
     ’limit’: ’600’, ’terms’: {’electric-based stores’: ’15’, ’
```

```
   Film-based stores’: ’10’}, ’operator’: ’LESS_OR_EQUAL’}], ’
   vars’: [’film-based’, ’electrical-based stores’]}
----------------------------------------
Predicted
{’obj_declaration’: {’type’: ’objvar’, ’direction’: ’minimize’, ’
   name’: ’total number of stores’, ’vars’: [’film-based’, ’
   electrical-based stores’]}, ’const_declarations’: [{’type’: ’
   lowerbound’, ’direction’: ’at least’, ’limit’: ’5’, ’var’: ’
   film-based stores’, ’operator’: ’GREATER_OR_EQUAL’}, {’type’:
   ’linear’, ’direction’: ’at least’, ’limit’: ’170’, ’terms’: {’
   Film-based stores’: ’2’, ’electrical-based store’: ’four’}, ’
   operator’: ’GREATER_OR_EQUAL’}, {’type’: ’linear’, ’direction
   ’: ’at most’, ’limit’: ’600’, ’terms’: {’Film-based stores’:
   ’10’, ’electric-based stores’: ’15’}, ’operator’: ’
   LESS_OR_EQUAL’}], ’vars’: [’film-based’, ’electrical-based
   stores’], ’id’: ’-1194187124’}
```

Swapped limits: The limits of the constraints gets swapped between two constraints.

```
Actual:
{’obj_declaration’: {’type’: ’objvar’, ’direction’: ’minimize’, ’
   name’: ’total number of shifts’, ’vars’: [’dentists’, ’oral
   hygienists’]}, ’const_declarations’: [{’type’: ’linear’, ’
   direction’: ’require’, ’limit’: ’1000’, ’terms’: {’Dentists’:
   ’12’, ’oral hygienists’: ’5’}, ’operator’: ’GREATER_OR_EQUAL
   ’}, {’type’: ’sum’, ’direction’: ’at least’, ’limit’: ’20’, ’
   operator’: ’GREATER_OR_EQUAL’}, {’type’: ’linear’, ’direction
   ’: ’budget’, ’limit’: ’65000’, ’terms’: {’dentists’: ’900’, ’
   oral hygienists’: ’250’}, ’operator’: ’LESS_OR_EQUAL’}], ’vars
   ’: [’dentists’, ’oral hygienists’]}
------------------------------------------------
Predicted
{’obj_declaration’: {’type’: ’objvar’, ’direction’: ’minimize’, ’
   name’: ’total number of shifts’, ’vars’: [’dentists’, ’oral
   hygienists’]}, ’const_declarations’: [{’type’: ’linear’, ’
   direction’: ’budget’, ’limit’: ’65000’, ’terms’: {’Dentists’:
   ’12’, ’oral hygienists’: ’5’}, ’operator’: ’LESS_OR_EQUAL’},
   {’type’: ’sum’, ’direction’: ’at least’, ’limit’: ’20’, ’
   operator’: ’GREATER_OR_EQUAL’}, {’type’: ’linear’, ’direction
   ’: ’require’, ’limit’: ’1000’, ’terms’: {’Dentists’: ’900’, ’
   oral hygienists’: ’250’}, ’operator’: ’GREATER_OR_EQUAL’}], ’
   vars’: [’dentists’, ’oral hygienists’]}
```

Syntax Errors Model hallucinates and produces incorrect literals which makes that beam invalid even though it has the right formulation;

```
num_of_constraints = 2
obj_declaration = {’type’: ’objvar’, ’direction’:’minimize’, ’name
   ’: ’number of branches’, ’vars’: [’urban branches’,’remote
   branch’]}
const_declarations = [{’type’: ’linear’, ’direction’: ’available’,
    ’limit’: ’550’, ’terms’: {’remote branch’: ’5’, ’urban branch
   ’: ’12’}, ’operator’: ’LESS_OR_EQUAL’}, {’type’: ’linear’, ’
   direction’: ’at least’, ’limit’: ’1200’, ’terms’: {’remote
   branch’: ’2’, ’urban branch’: ’7’}, ’operator’: ’
   GREATER_OR_EQUAL’}]
 yield’}]
```

7.4 POST PROCESSING RULES - SUBSET

Post model Corrections:

```
1. If a Linear objective's terms have same coefficients, convert
   the objective into a 'objvar' type
2. If the same coefficients are being reused in the constraints,
   most probably it is a 'objvar' type.
3. If the constraint type is "lowerbound" the operator should be "
   GREATER_THAN_EQUAL_TO" and vice versa
4. if the constraint type is "sum" then there should be no terms
   allowed for the variables (i.e coeff of all variables is 1)
```

Beam Search scoring:

```
Violation of each rule reduces the corresponding beam's score:

1. There should be at least as many constraints as the constraint
   directions available in the SPANs from tagged input.
2. All the limits ( RHS of the inequality ) must be available in
   the formulation, proportionate penalty will be imposed on the
   total score
3. All the parameters ( coefficients in LHS ) should be available
   in the formulation, penalty will be imposed proportionately.
4. The scale difference of the coefficients for a given constraint
    has to be below the maximum threshold. Proportionate penalty
   to be applied on the total score.
5. If there is a postive constraint on the decision variables, the
    constraints must be positive as well. Meaning increasing the
   parameter value should also increase the limit.
```

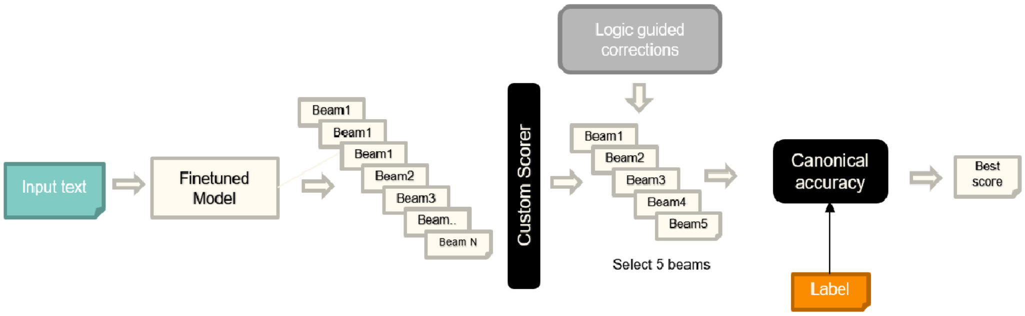
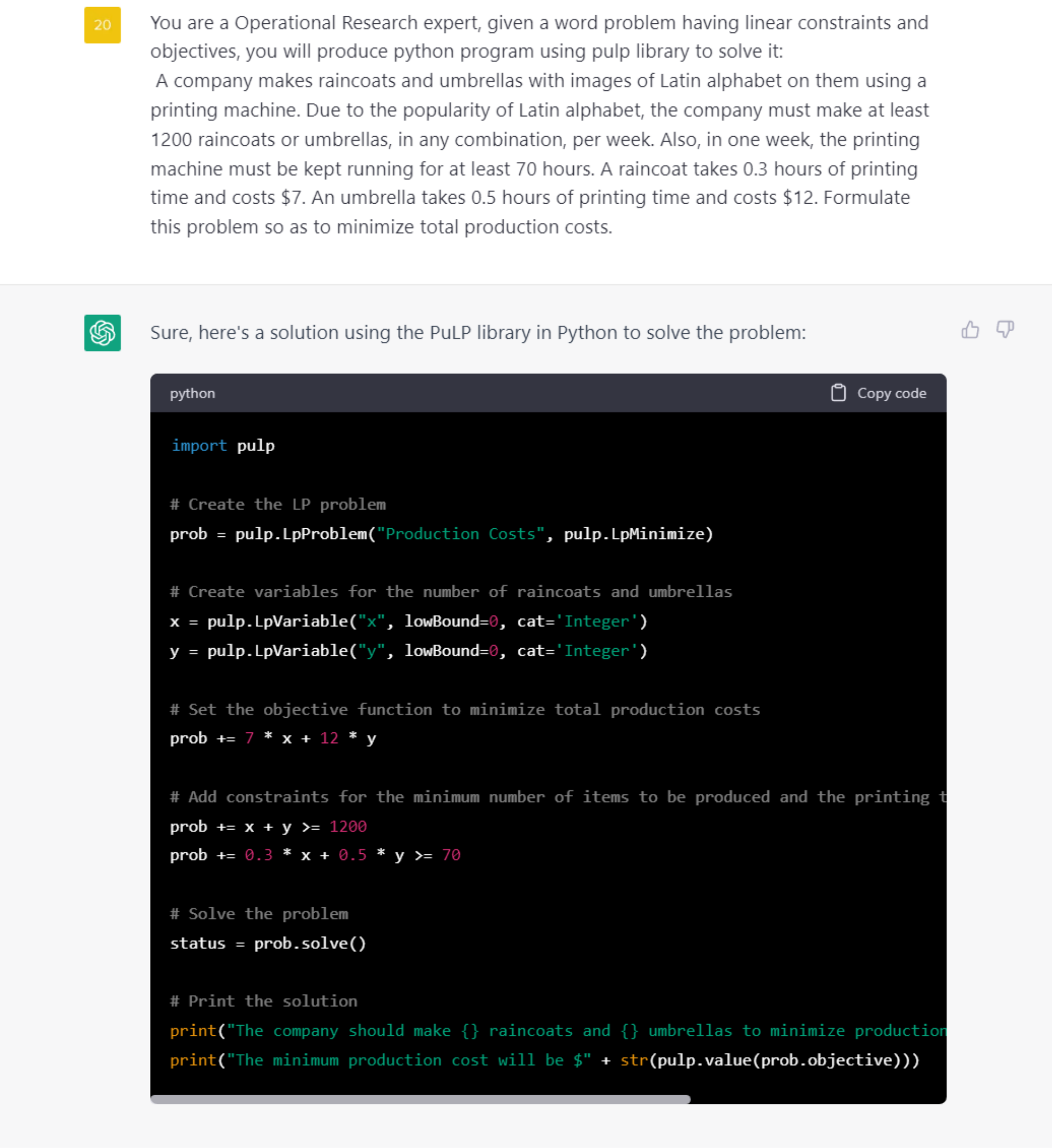


Figure 3: Candidate selection process

7.5 CHATGPT EXPERIMENTS

Since official API for ChatGPT was not available at the time of writing this paper, we had to rely on manual methods to prompt ChatGPT. We prompted with a standard prompt for all the problems. Copied the response from ChatGPT and run it without any modification. Refer Figure 4 and 5. We found few examples where the use of pulp variables were wrong and resulted in a Exception. Such examples where marked with a 0 for optimal value.

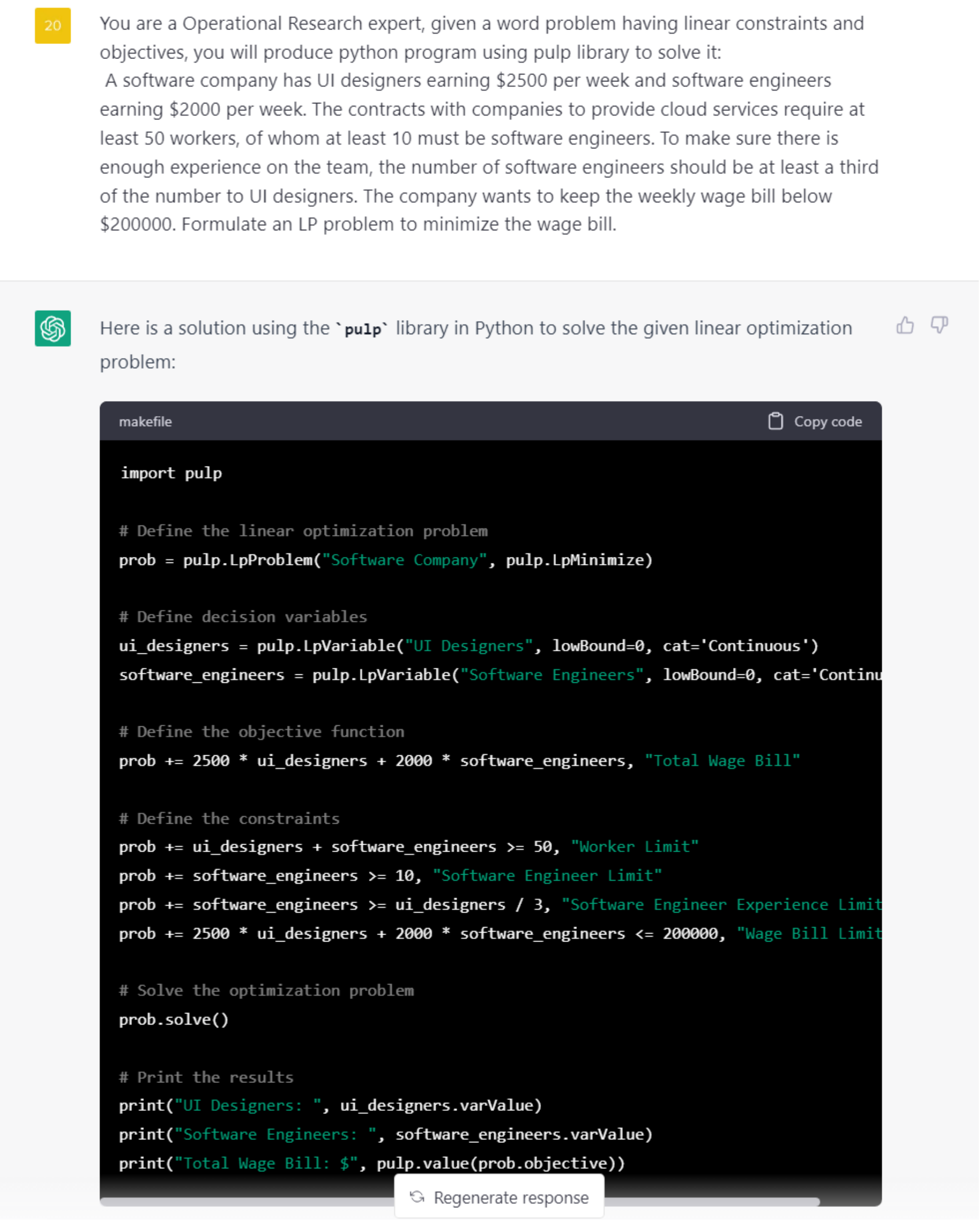


You are a Operational Research expert, given a word problem having linear constraints and objectives, you will produce python program using pulp library to solve it:
A company makes raincoats and umbrellas with images of Latin alphabet on them using a printing machine. Due to the popularity of Latin alphabet, the company must make at least 1200 raincoats or umbrellas, in any combination, per week. Also, in one week, the printing machine must be kept running for at least 70 hours. A raincoat takes 0.3 hours of printing time and costs $7. An umbrella takes 0.5 hours of printing time and costs $12. Formulate this problem so as to minimize total production costs.

Sure, here's a solution using the PuLP library in Python to solve the problem:

```python
import pulp

# Create the LP problem
prob = pulp.LpProblem("Production Costs", pulp.LpMinimize)

# Create variables for the number of raincoats and umbrellas
x = pulp.LpVariable("x", lowBound=0, cat='Integer')
y = pulp.LpVariable("y", lowBound=0, cat='Integer')

# Set the objective function to minimize total production costs
prob += 7 * x + 12 * y

# Add constraints for the minimum number of items to be produced and the printing t
prob += x + y >= 1200
prob += 0.3 * x + 0.5 * y >= 70

# Solve the problem
status = prob.solve()

# Print the solution
print("The company should make {} raincoats and {} umbrellas to minimize production
print("The minimum production cost will be $" + str(pulp.value(prob.objective)))
```

Figure 4: ChatGPT - Question 1



You are a Operational Research expert, given a word problem having linear constraints and objectives, you will produce python program using pulp library to solve it:
A software company has UI designers earning $2500 per week and software engineers earning $2000 per week. The contracts with companies to provide cloud services require at least 50 workers, of whom at least 10 must be software engineers. To make sure there is enough experience on the team, the number of software engineers should be at least a third of the number to UI designers. The company wants to keep the weekly wage bill below $200000. Formulate an LP problem to minimize the wage bill.

Here is a solution using the `pulp` library in Python to solve the given linear optimization problem:

```
import pulp

# Define the linear optimization problem
prob = pulp.LpProblem("Software Company", pulp.LpMinimize)

# Define decision variables
ui_designers = pulp.LpVariable("UI Designers", lowBound=0, cat='Continuous')
software_engineers = pulp.LpVariable("Software Engineers", lowBound=0, cat='Continu

# Define the objective function
prob += 2500 * ui_designers + 2000 * software_engineers, "Total Wage Bill"

# Define the constraints
prob += ui_designers + software_engineers >= 50, "Worker Limit"
prob += software_engineers >= 10, "Software Engineer Limit"
prob += software_engineers >= ui_designers / 3, "Software Engineer Experience Limit
prob += 2500 * ui_designers + 2000 * software_engineers <= 200000, "Wage Bill Limit

# Solve the optimization problem
prob.solve()

# Print the results
print("UI Designers: ", ui_designers.varValue)
print("Software Engineers: ", software_engineers.varValue)
print("Total Wage Bill: $", pulp.value(prob.objective))
```

Figure 5: ChatGPT - Question 2