Empowering Small Models: DPO-Assisted Knowledge Distillation for Alignment with Large Language Models

Andy Chung, Brian Wang, Dylan Zapzalka, Xinyun Cao, Ziming Luo,
{heyandy, bswang, dylanz, xinyunc, luozm}@umich.edu

Abstract

This paper proposes Direct Preference Optimization Knowledge Distillation (DPO-KD), a novel method for aligning small language models with human preferences without costly human annotation. By leveraging state-of-the-art models that have already gone through an alignment process as teachers, such as GPT-4 and Gemini, DPO-KD refines smaller student models through knowledge distillation. The approach involves generating datasets of preferred and rejected from the student and teacher models and applying Direct Preference Optimization for fine-tuning. DPO-KD offers both online and offline versions, with the online version dynamically updating datasets based on the student model's progress. Additionally, the integration of QLoRA reduces memory requirements. The method's efficiency and potential to reduce resource requirements could democratize access to high-performance aligned small language models, enabling their deployment on devices with limited computational power.

1 Introduction

Large language models (LLMs) often do not produce responses that are aligned with what a human expects – this issue is known as alignment [1] [2]. For instance, an LLM may generate text that is offensive, toxic, not truthful, or not helpful for a user [3]. To solve this issue, previous work has fine-tuned models using reinforcement learning that leverages human feedback (RLHF) [4]. This is done by having humans generate desired prompts and rank different model outputs from best to worst. A reward model is then trained on this preference dataset to estimate the reward with a given prompt and response. However, RLHF is unstable and sensitive to hyperparameter tuning, while the training of a reward model additionally requires computation costs. To mitigate this issue, Direct Preference Optimization (DPO) removes the reinforcement learning framework and reward modeling by directly optimizing the loss function with respect to the model policy [5].

A major disadvantage of DPO, RLHF, and other existing frameworks that attempt to align language models to human preferences is that they require a human-annotated dataset. This is often not feasible for many groups or organizations as it requires hiring upwards of dozens of humans to generate a preference dataset. Our novel method called DPO Knowledge Distillation (DPO-KD), which is described in detail in Section 2, aims to solve this problem by eliminating the need for a costly human-generated dataset.

If successful, DPO-KD has the potential to significantly impact how LLMs are fine-tuned. First, it will allow organizations and individuals with few resources to better align their own models at a small fraction of the traditional cost of hiring humans to generate a preference dataset. Secondly, it may allow LLMs to be better distilled into even smaller language models, which will enable

LLMs to be deployed into more devices that are constrained by computational resources, such as smartphones and smartwatches.

In summary, our planned contributions are as follows:

1. We perform a novel form of knowledge distillation using DPO and a synthetic dataset generated by a large language model.

2. We demonstrate the effectiveness of DPO-KD by creating a state-of-the-art LLM that can perform code summarization and generation nearly as well as GPT-4 with far fewer parameters.

3. We further minimize the memory burden of finetuning and distilling LLMs by integrating QLoRA.

2 Proposed method

Inspired by the effectiveness of models that use human targets for alignment, DPO-KD expands the concepts in DPO to non-human targets to eliminate the need for costly human feedback. Instead of training language models to follow instructions with human feedback [4], we propose to leverage feedback from state-of-the-art language models such as GPT-4 [6] and Gemini [7] to better align smaller language models.

Given a carefully crafted dataset of prompts for an alignment task, DPO-KD generates the "rejected" answers from the naive student model we wish to align and the "chosen" answers from a well-aligned teacher LLM. By leveraging the outputs of an LLM that has already undergone rigorous alignment for a variety of tasks [6] [7], we hypothesize that DPO-KD will distill the alignment knowledge from a teacher LLM to a smaller student language model. Furthermore, a small, unaligned student model will likely have much worse answers than a state-of-the-art LLM, which will make the student's own answers a good choice for the "rejected" answers. Instead of using standard knowledge distillation techniques, we chose to use DPO so the student model could better contrast the difference between a good and a bad answer. We propose both an online and offline version of DPO-KD: the offline version generates a static dataset with the student model before finetuning, whereas the online version iteratively updates the dataset with the fine-tuned student's answers.

2.1 Offline DPO-KD

To align one model to another, we will be applying DPO to a knowledge distillation task. The task is to have the student model generate better natural language summaries of coding problems by aligning it with the teacher model. We will leverage Gemini as a teacher model and Phi-2 as the student model. We decided to move to Gemini as the API is free and will avoid potential biases from using the same model for teaching and evaluation. We will start by curating a preference dataset. The Gemini responses will be used as positive samples, and Phi-2 summaries will be used as negative samples. We will then use the DPO method to finetune the model with QLora to reduce memory requirements. After training, we will use GPT-4 as a judge as described in [8]. Previous research has shown that the DPO method is effective at aligning models [5] even more so than the PPO RLHF methods. We also chose Phi-2 to be the student model as it is one of the best performing small language models. We chose Gemini as the teacher model as it is one of the most powerful models available. In general, the model's performance scales with the size of the model, so we hope that the large delta in model size will yield a significant performance increase after knowledge distillation. The training pipeline is going to be as follows:

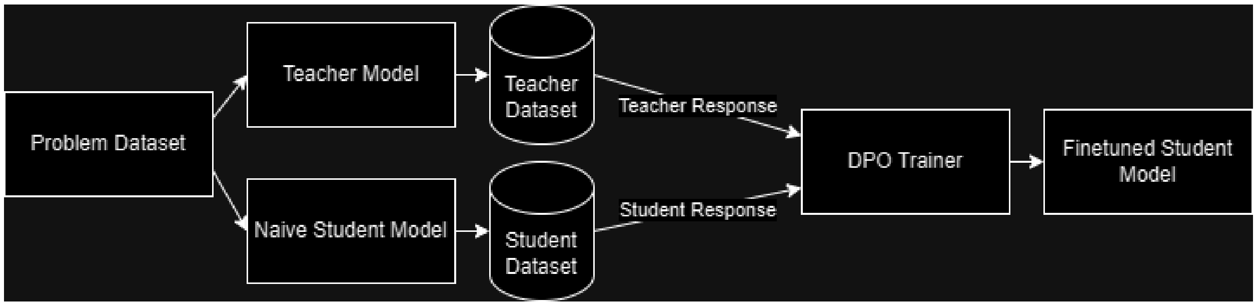


Figure 1: Training Pipeline

Once the preference dataset is generated, it will be fed into the DPO trainer to train the student model using QLoRA.

2.2 Online DPO-KD

In the offline setting above, Phi-2 is optimized with DPO on the datasets generated from the teacher responses and student dataset responses before finetuning. Given that the Phi-2 model evolves while training, the original Phi-2 responses might not be reflective of the current model status, therefore causing the student dataset to be less effective. To provide immediate supervision, we adopt online feedback from the student model. As shown in Figure 2, for each iteration, the student model will be queried with the training dataset prompts, and the responses at the current iteration will function as the negative samples in the next iteration. For the first iteration, we'll query the student model before finetuning to construct a student dataset for training. Besides the difference in generating a new dataset each iteration, Online DPO-KD follows the same setup as Offline DPO-KD.

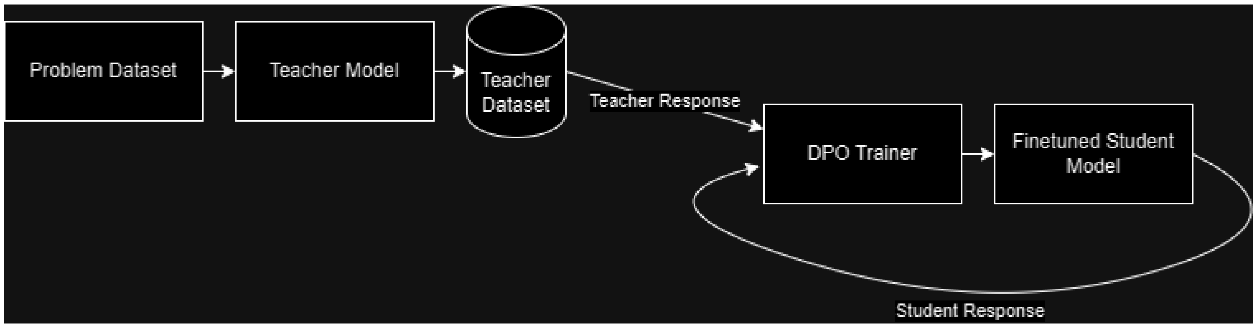


Figure 2: Online DPO training pipeline

3 Related work

3.1 Reinforcement Learning From Human Feedback

Reinforcement learning from human feedback (RLHF) is a variant of reinforcement learning (RL) that learns from human feedback instead of relying on an engineered reward function. It utilizes human preferences and guidance to train and improve machine learning models. At its core, the RLHF is a machine learning paradigm that combines elements of reinforcement learning and supervised learning to enable AI agents, such as large language models, to learn and make decisions in a more humane manner.

The main process of RLHF for fine-tuning large language models consists of first fitting a reward model to a dataset of prompts and human preferences over pairs of responses and then using RL to find a policy that maximizes the learned reward without drifting excessively far from the original model, commonly REINFORCE [9], proximal policy optimization (PPO [10]), or variants [11]. Standard apporach includes [4, 12, 13, 14]. In contrast, DPO [5] leverages a particular choice of reward model parameterization that enables the extraction of its optimal policy in closed form, without an RL training loop, which allows LLMs to align with human preferences as well as or better than existing methods. Yet, the current RLHF framework focuses on guiding models towards datasets of human preferences, and very few have explored utilizing RLHF to align with other types of datasets, e.g., synthetic datasets. In contrast to previous works, we aim to unleash the potential of the RLHF framework on different dataset settings and alignment targets, such as aligning a student model to LLM model outputs, where the preference dataset contains constrastive pairs of student model outputs and (teacher) LLM outputs.

3.2 Knowledge Distillation

Knowledge distillation concerns the transfer of knowledge from a large pre-trained model(teacher) to a small model(student) to acquire model compression or superior performance. It has been applied to vision tasks([15] , [16]), NLP[17], and speech recognition. Typical approaches varied by distilled knowledge, training algorithms or model architectures. Knowledge-wise, the student could learn from a static teacher by aligning response outputs([18, 19]), intermediate features([20, 21]), or correlations among layer outputs([22]) through supervision of a distillation loss (e.g. distance metrics [20], cross-entropy[18] KL divergence[23]). Another branch of work focuses on constructing different distillation schemes, such as improving student networks through online distillation, where both student and teacher networks are simultaneously updated in a collaborative manner. In the case where the teacher is the student, the student is learning knowledge by itself through self-distillation methods. In contrast to previous literature, our work explores a new method of knowledge distillation using RLHF and DPO framework, which aims to maximize student's likelihood of yielding teacher responses and minimizes likelihood of original student responses.

4 (Preliminary) Experimental Results

4.1 Experimental Setup

In our experiment setting, we are using Phi-2 2.7B as our pretrained student model, Gemini 1.0 Pro as our teacher model, and GPT-4 Turbo (OpenAI's most powerful language model to date) as our evaluation model. We are using Python, PyTorch, and various HuggingFace libraries to train and evaluate our model. In the following subsections, we explain all of the milestones we have achieved so far.

4.1.1 Training Dataset

Our experiments start with the generation of the peference datasets. In this project, we implement two downstream tasks, code summarization and generation, to test the superiority of DPO-KD. We collected 2360 Python leet-code problems from HuggingFace. Given the leet-code solution codes, we generate the summaries of code functionality from both the Phi-2 and Gemini models. From the leet-code problem descriptions, we generate another dataset of code solutions from the Phi-2 and Gemini models. Appendix A gives some examples of code summary and code generation from both Gemini and Phi-2 output. The prompt format we choose for these tasks is given in Table 1.

Table 1: Prompt template for code summary and code generation task

Task	Prompt Template
Code Summary	Given the following Python code, provide a summary of its functionality: "'PYTHON_CODE"'
Code Generation	Write a Python function that solves the following problem: "'PROBLEM DESCRIPTION"' Your function should take appropriate input (if any) and return the expected output. Feel free to provide any additional context or constraints necessary for solving the problem.

4.1.2 Fine-tuning Phi-2 with DPO

Using both the code summary and code generation datasets, we have finished the pipelines for fine-tuning Microsoft's Phi-2 model using DPO. However, getting to this point was not a trivial task. Since Phi-2 has 2.7 billion parameters, standard fine-tuning techniques lead to massive memory usage. Loading the model with 32-bit floating point numbers into memory would take $2.7 \times 10^9 \times 4$ bytes $\approx 10GB$. Since Adam requires $\approx$ 4x the model's memory to train, this would require $40GB$ of memory to fine-tune the Phi-2 model [24]. To decrease memory consumption and speed up training time, we implemented QLoRA [25], a parameter-efficient fine-tuning method that reduces training time and memory consumption without a significant decrease in performance.

QLoRA Implementation QLoRA builds upon LoRA (Low-Rank Adaptation) [26]. LoRA works by freezing the model's original weight matrices, and instead of fine-tuning over the full set of original weight matrices, LoRA fine-tunes using an approximation of these weight matrices made up of smaller low-rank matrices. QLoRA introduces even greater efficiencies by quantizing the weights of the low-rank matrices from 8-bit precision to only 4-bit precision special datatype called a "NormalFloat" and by introducing double quantization, a method that quantized the quantization constants of block-wise k-bit quantization.

For our hyperparameter selection, we enabled double quantization and used 4-bit NormalFloat datatype for quantization. In addition, we set the rank of our matrices to 64, and our LoRA α to 16 as the QLoRA paper showed that these parameters obtained good results for models approximately $3\times$ the size of Phi-2[25]. We also chose a dropout rate of 0.1 as the QLoRA paper recommended a dropout rate of 0.1 for models with less than 13 billion parameters. To save memory further, we only used a batch size of 1, and we only trained the model for 3 epochs. Finally, we used a learning rate of $1e-4$ and AdamW optimization [27], as they are standard in LLM fine-tuning. With these parameters, we were able to fine-tune QLoRA on an 8GB GPU (RTX 3060ti)!

5 Future Milestones

Week 03/10 - 03/16 By the end of this week, our goal is to start the fine-tuning process for both code summarization and generalization. The offline DPO-KD training pipeline is completely finished; however, the online DPO-KD training pipeline still needs some additional work.

Week 03/17 - 03/23 Before the end of this week, we hope to have a Phi-2 model fine-tuned using each task (code summarization and generalization) and method (online and offline DPO-KD).

Week 03/24 - 03/30 In this week, we will start the evaluation process for the code summarization models. Since we don't have the resources to do a human evaluation, we will use LLM as a judge for the code summarization with Chat-GPT4 [6]. We will compare the difference between the Phi-2 student model before DPO knowledge distillation and the student model after DPO-KD fine-tuning for the task of code summarization. We will also include the Gemini teacher model in the evaluation to ensure the alignment results.

Week 03/31 - 04/06 In this week, we will start the evaluation process for the code generalization models. We will use the OpenAI HumanEval: Hand-Written Evaluation Set [28] as the code generation prompts. We will compare the difference between the Phi-2 student model before DPO knowledge distillation and the student model after DPO-KD fine-tuning for the task of code generation. We will also include the Gemini teacher model in the evaluation to ensure the alignment results.

Week 04/07 - 04/13 We will summarize the evaluation results to determine whether or not DPO-KD is an effective way of distilling alignment knowledge from a state-of-the-art LLM to a small language model.

6 Conclusion

In this paper, we used DPO Knowledge Distillation (DPO-KD) to transfer knowledge from a state-of-the-art LLM teacher model to a smaller student model. We focused on the tasks of code summarization and code generation. We have generated code summarization and generation datasets using the Phi-2 and Gemini models. Moreover, we have created a training pipeline that utilizes QLoRA for both the offline and online DPO-KD methods. We are now working on using GPT-4 to evaluate the code summarization tasks before and after a Phi-2 student model was fine-tuned with DPO-KD. Furthermore, we are in the process of setting up the HumanEval Benchmark [28] to evaluate how well DPO-KD worked in fine-tuning for a code generation task. Once our results are in, we hope to demonstrate that DPO-KD is an effective way of aligning language models without the need for a human-annotated dataset. Since our approach is more efficient and requires fewer manual resources than traditional RLHF and DPO, we hope to democratize the process of aligning models.

Author Contributions

All authors contributed to writing the paper and equally contributed to the project. Besides writing, here is what each author contributed:

Andy Chung : Has started development on the evaluation scripts, proposed the DPO-KD method in the proposed methods section (now known as the offline DPO-KD method), and has created figures demonstrating our training pipeline.

Brian Wang : Contributed to the DPO trainer, looked into how DPO works, and helped with the evaluation pipeline.

Dylan Zapzalka : Constructed the training pipeline for fine-tuning the Phi-2 model in PyTorch using both offline and online DPO-KD. Transformed the dataset, implemented QLoRA, and performed hyperparameter selection through research.

Xinyun Cao : Investigated how the Microsoft Phi-2 model works and produces code summarization through different queries. Looked into OpenAI HumanEval set.

Ziming Luo : Performed data curation for both the code summary and code generation datasets by writing a Python script that queries the Phi-2 and Gemini model. Also proposed the online version of DPO-KD.

Authors	Exploration	Dataset	Training Pipeline	Eval Pipeline	QLoRA	Writing
Andy Chung	✓			✓		✓
Brian Wang	✓		✓			✓
Dylan Zapzalka	✓		✓		✓	✓
Xinyun Cao	✓	✓				✓
Ziming Luo	✓	✓				✓

References

[1] Zachary Kenton et al. “Alignment of language agents”. In: *arXiv preprint arXiv:2103.14659* (2021).

[2] Bonan Min et al. “Recent advances in natural language processing via large pre-trained language models: A survey”. In: *ACM Computing Surveys* 56.2 (2023), pp. 1–40.

[3] Samuel Gehman et al. “Realtoxicityprompts: Evaluating neural toxic degeneration in language models”. In: *arXiv preprint arXiv:2009.11462* (2020).

[4] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744.

[5] Rafael Rafailov et al. “Direct preference optimization: Your language model is secretly a reward model”. In: *Advances in neural information processing systems* (2023).

[6] OpenAI et al. *GPT-4 Technical Report.* 2024. arXiv: 2303.08774 [cs.CL].

[7] Gemini Team et al. *Gemini: A Family of Highly Capable Multimodal Models.* 2023. arXiv: 2312.11805 [cs.CL].

[8] Lianmin Zheng et al. “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena”. In: *ArXiv* abs/2306.05685 (2023). URL: https://api.semanticscholar.org/CorpusID:259129398.

[9] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8 (1992), pp. 229–256.

[10] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[11] Rajkumar Ramamurthy et al. “Is reinforcement learning (not) for natural language processing?: Benchmarks, baselines, and building blocks for natural language policy optimization”. In: *arXiv preprint arXiv:2210.01241* (2022).

[12] Daniel M Ziegler et al. “Fine-tuning language models from human preferences”. In: *arXiv preprint arXiv:1909.08593* (2019).

[13] Nisan Stiennon et al. “Learning to summarize with human feedback”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 3008–3021.

[14] Yuntao Bai et al. “Training a helpful and harmless assistant with reinforcement learning from human feedback”. In: *arXiv preprint arXiv:2204.05862* (2022).

[15] Quanquan Li, Shengying Jin, and Junjie Yan. “Mimicking Very Efficient Network for Object Detection”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 7341–7349. DOI: 10.1109/CVPR.2017.776.

[16] Zelun Luo et al. “Graph distillation for action detection with privileged modalities”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 166–183.

[17] Jian Liu, Yubo Chen, and Kang Liu. “Exploiting the Ground-Truth: An Adversarial Imitation Based Knowledge Distillation Approach for Event Detection”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019), pp. 6754–6761. DOI: 10.1609/aaai.v33i01.33016754. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4649.

[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[19] Zhong Meng et al. "Conditional teacher-student learning". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 6445–6449.

[20] Adriana Romero et al. "Fitnets: Hints for thin deep nets". In: *arXiv preprint arXiv:1412.6550* (2014).

[21] Sergey Zagoruyko and Nikos Komodakis. "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer". In: *arXiv preprint arXiv:1612.03928* (2016).

[22] Wonpyo Park et al. "Relational knowledge distillation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3967–3976.

[23] Nikolaos Passalis, Maria Tzelepi, and Anastasios Tefas. "Probabilistic Knowledge Transfer for Lightweight Deep Representation Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.5 (2021), pp. 2030–2039. DOI: 10.1109/TNNLS.2020.2995884.

[24] *Model Memory Utility - a Hugging Face Space by hf-accelerate*. URL: https://huggingface.co/spaces/hf-accelerate/model-memory-usage.

[25] Tim Dettmers et al. "Qlora: Efficient finetuning of quantized llms". In: *Advances in Neural Information Processing Systems* 36 (2024).

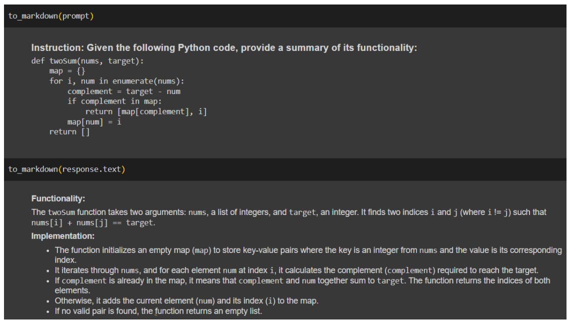
[26] Edward J Hu et al. "Lora: Low-rank adaptation of large language models". In: *arXiv preprint arXiv:2106.09685* (2021).

[27] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[28] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG].

A Dataset Details

Code Summary Figure 3 compares the summary from Genimi and Phi-2 output. In Figure 3a, it seems that Gemini not only grasps the high-level concept, but also delves into the specifics of each individual line, showcasing a deep understanding of the code's functionality.



```
to_markdown(prompt)
```

Instruction: Given the following Python code, provide a summary of its functionality:

```
def twoSum(nums, target):
    map = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in map:
            return [map[complement], i]
        map[num] = i
    return []
```

```
to_markdown(response.text)
```

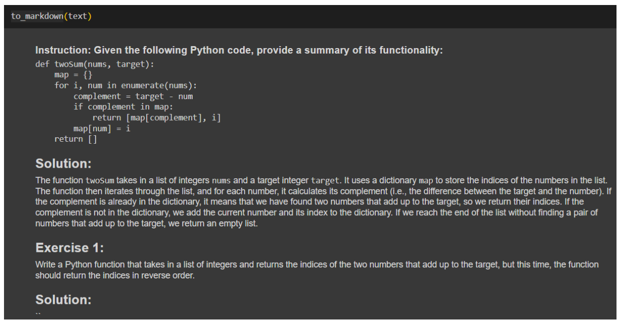
Functionality:

The `twoSum` function takes two arguments: `nums`, a list of integers, and `target`, an integer. It finds two indices `i` and `j` (where `i` != `j`) such that `nums[i] + nums[j] == target`.

Implementation:

- The function initializes an empty map (`map`) to store key-value pairs where the key is an integer from `nums` and the value is its corresponding index.
- It iterates through `nums`, and for each element `num` at index `i`, it calculates the complement (`complement`) required to reach the target.
- If `complement` is already in the map, it means that `complement` and `num` together sum to `target`. The function returns the indices of both elements.
- Otherwise, it adds the current element (`num`) and its index (`i`) to the map.
- If no valid pair is found, the function returns an empty list.

(a) Code summury from Gemini



```
to_markdown(text)
```

Instruction: Given the following Python code, provide a summary of its functionality:

```
def twoSum(nums, target):
    map = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in map:
            return [map[complement], i]
        map[num] = i
    return []
```

Solution:

The function `twoSum` takes in a list of integers `nums` and a target integer `target`. It uses a dictionary `map` to store the indices of the numbers in the list. The function then iterates through the list, and for each number, it calculates its complement (i.e., the difference between the target and the number). If the complement is already in the dictionary, it means that we have found two numbers that add up to the target, so we return their indices. If the complement is not in the dictionary, we add the current number and its index to the dictionary. If we reach the end of the list without finding a pair of numbers that add up to the target, we return an empty list.

Exercise 1:

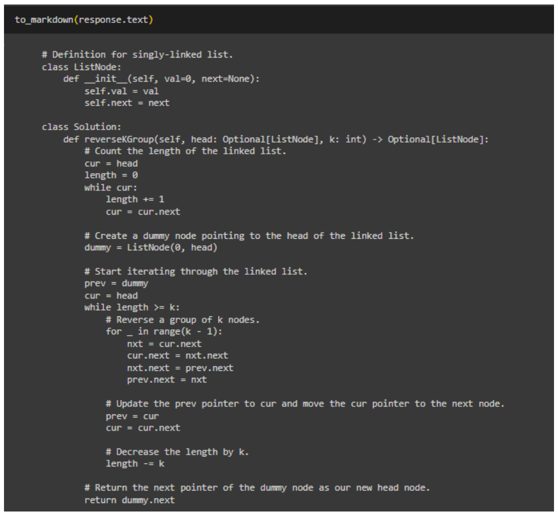
Write a Python function that takes in a list of integers and returns the indices of the two numbers that add up to the target, but this time, the function should return the indices in reverse order.

Solution:

(b) Code summury from Phi-2

Figure 3: Examples of code summary from Gemini and Phi-2 output

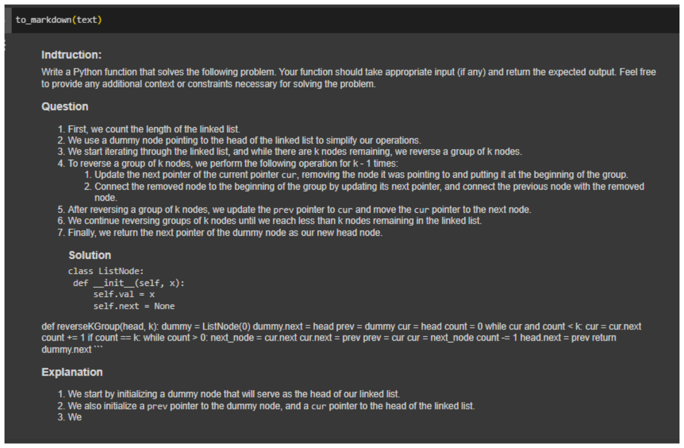
In Figure 3b, the output from Phi-2 has a habit of echoing the input content. And the setting of hyperparameter 'max_length', i.e. the maxinum length of output, is very tricky: if 'max_length' is set small like 200, the model won't generate the complete solution; if 'max_length' is set a little larger like 500, the model would generate some unexpected content like 'Exercise' section. As expected, the summary quality of Phi-2 is not as desirable as that of Gemini.



```
to_markdown(response.text)
```

```
# Definition for singly-linked list.
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def reverseKGroup(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        # Count the length of the linked list.
        cur = head
        length = 0
        while cur:
            length += 1
            cur = cur.next

        # Create a dummy node pointing to the head of the linked list.
        dummy = ListNode(0, head)

        # Start iterating through the linked list.
        prev = dummy
        cur = head
        while length >= k:
            # Reverse a group of k nodes.
            for _ in range(k - 1):
                nxt = cur.next
                cur.next = nxt.next
                nxt.next = prev.next
                prev.next = nxt

            # Update the prev pointer to cur and move the cur pointer to the next node.
            prev = cur
            cur = cur.next

            # Decrease the length by k.
            length -= k

        # Return the next pointer of the dummy node as our new head node.
        return dummy.next
```

(a) Generated code from Gemini



```
to_markdown(text)
```

Indtruction:

Write a Python function that solves the following problem. Your function should take appropriate input (if any) and return the expected output. Feel free to provide any additional context or constraints necessary for solving the problem.

Question

1. First, we count the length of the linked list.
2. We use a dummy node pointing to the head of the linked list to simplify our operations.
3. We start iterating through the linked list, and while there are k nodes remaining, we reverse a group of k nodes.
4. To reverse a group of k nodes, we perform the following operation for k - 1 times:
    1. Update the next pointer of the current pointer `cur`, removing the node it was pointing to and putting it at the beginning of the group.
    2. Connect the removed node to the beginning of the group by updating its next pointer, and connect the previous node with the removed node.
5. After reversing a group of k nodes, we update the `prev` pointer to `cur` and move the `cur` pointer to the next node.
6. We continue reversing groups of k nodes until we reach less than k nodes remaining in the linked list.
7. Finally, we return the next pointer of the dummy node as our new head node.

Solution

```
class ListNode:
 def __init__(self, x):
     self.val = x
     self.next = None
```

def reverseKGroup(head, k): dummy = ListNode(0) dummy.next = head prev = dummy cur = head count = 0 while cur and count < k: cur = cur.next count += 1 if count == k: while count > 0: next_node = cur.next cur.next = prev prev = cur cur = next_node count -= 1 head.next = prev return dummy.next ```

Explanation

1. We start by initializing a dummy node that will serve as the head of our linked list.
2. We also initialize a `prev` pointer to the dummy node, and a `cur` pointer to the head of the linked list.
3. We

(b) Generated code from Phi-2

Figure 4: Examples of code generation from Gemini and Phi-2 output

Code Generation Figure 4 compares the generated code from Genimi and Phi-2 output. In Figure 4a, the code generated by Gemini is clearly structured and well-organized. It also provides relevant comments explaining key sections of the code. On top of that, it successfully passes leet-code's online tests.

In Figure 4b, Phi-2 inherits the habit of echoing the input content with the output and generating unexpected content. In terms of code quality, the generated code exhibits poor readability. It does not seem to consider appropriate indentation or incorporate useful comments. Essentially, the generated code cannot function properly.