CoLLiE: Collaborative Training of Large Language Models in an Efficient Way
Kai Lv†‡, Shuo Zhang†‡, Tianle Gu∗, Shuhao Xing1, Jiawei Hong1, Keyu Chen1
Xiaoran Liu1, Yuqing Yang1, Honglin Guo1, Tengxiao Liu1, Yu Sun1
Qipeng Guo1, Hang Yan2, Xipeng Qiu‡
1 School of Computer Science, Fudan University, 2 Shanghai AI Laboratory
1{klv21, szhang22, hongjw21, kychen22, liuxr22}@m.fudan.edu.cn
2{shxing, hlguo20, qpqiu16, xpqiu}@fudan.edu.cn
1{gutianle, yanhang}@pjlab.org.cn

Abstract
Large language models (LLMs) are increasingly pivotal in a wide range of natural language processing tasks. Access to pre-trained models, courtesy of the open-source community, has made it possible to adapt these models to specific applications for enhanced performance. However, the substantial resources required for training these models necessitate efficient solutions. This paper introduces CoLLiE, an efficient library that facilitates collaborative training of large language models using 3D parallelism, parameter-efficient fine-tuning (PEFT) methods, and optimizers such as Lion, Adan, Sophia, and LOMO. With its modular design and comprehensive functionality, CoLLiE offers a balanced blend of efficiency, ease of use, and customization. CoLLiE has proven superior training efficiency in comparison with prevalent solutions in pre-training and fine-tuning scenarios. Furthermore, we provide an empirical evaluation of the correlation between model size and GPU memory consumption under different optimization methods, as well as an analysis of the throughput. Lastly, we carry out a comprehensive comparison of various optimizers and PEFT methods within the instruction-tuning context. CoLLiE is available at https://github.com/OpenLMLab/collie.

Figure 1: The two stages of training pre-trained language models, during which CoLLiE exhibits efficiency.

1 Introduction
Large language models (LLMs) have demonstrated remarkable abilities across various natural language processing tasks and showcased potential as intelligent assistants. Thanks to the vibrant open-source community, multiple excellent large language models’ weights are accessible, including OPT (Zhang et al., 2022), BLOOM (Scao et al., 2022), LLaMA (Touvron et al., 2023), etc. Despite the impressive general capabilities of pre-trained LLMs, training for particular application scenarios can lead to even more outstanding performance. As shown in Figure 1, the training process can be divided into two stages: 1. Further pre-training, which supplements specific domain knowledge and expands the vocabulary to enhance tokenization efficiency; 2. Instruction-tuning, which adapts the model to downstream tasks and improves its instruction-following ability.

With the scaling of language models, the resources required for training have increased substantially, making it infeasible to train the entire model on a single GPU. Model parallelism addresses this issue by partitioning the model across different GPUs, distributing the training workload among these GPUs. This can be achieved through three methods: tensor parallelism (TP, Shoeybi et al. (2019)), pipeline parallelism (PP, Huang et al. (2019); Narayanan et al. (2019)), and stage 3 of zero redundancy optimizer (ZeRO-3, Rajbhandari et al. (2020)). In addition, during the instruction-tuning stage, there are resource-efficiency and training-effectiveness trade-off approaches (Sun et al., 2023b): parameter-efficient fine-tuning (PEFT) methods (Ding et al., 2023). These methods selectively choose or add a few parameters for training, effectively reducing the GPU memory required to train large language models.

In this context, we introduce CoLLiE, an easy-to-use library for Collaborative training of Large Language Models in an Efficient way. The library not only integrates the previously mentioned three parallelism strategies and PEFT methods, but also...
implements efficient optimizers such as Lion (Chen et al., 2023), Adan (Xie et al., 2022), Sophia (Liu et al., 2023), and LOMO (Lv et al., 2023). We have restructured multiple mainstream open-source models to support TP and PP and incorporated FlashAttention (Diao et al., 2022; Dao, 2023) to further boost efficiency, while retaining interfaces similar to HuggingFace models within CollieModel class. Training efficiency is one of the most distinctive feature of CoLLiE, boasting a significantly higher training throughput compared to current popular solutions. CoLLiE also offers a wide range of functionalities, including data preprocessing, model training, checkpoint saving and monitoring and evaluation during training process. CoLLiE’s modular design allows for flexible combinations of parallelism strategies, PEFT methods, and training hyperparameters, which can be configured simply by modifying the CollieConfig class. Furthermore, CoLLiE is purposefully designed with extensibility, providing customizable functionalities. In summary, CoLLiE offers a comprehensive solution that caters to the needs of both beginners and experienced professionals. Our contributions can be summarized as follows:

- We introduce CoLLiE, an efficient and easy-to-use library for collaborative training of large language models.
- We empirically provide the relationship between model size and the actual GPU memory consumption using different optimization methods in real training scenarios.
- We compared the throughput of CoLLiE and the current prevailing solutions in (further) pre-training and fine-tuning scenarios, and CoLLiE demonstrates higher efficiency.
- We conducted a comprehensive comparison of different optimizers and PEFT methods in the context of instruction-tuning.

2 Background

PEFT Methods There has been a rise in using parameter-efficient fine-tuning (PEFT) techniques to adapt models for instruction-tuning by adjusting partial parameters. One of the early success is adapter tuning (Houlsby et al., 2019), which inserts trainable neural modules into transformers layers while keeping the original model unchanged. In line with adapter tuning, LoRA (Hu et al., 2022) reparameterizes the dense layers and only updates low rank matrices while introducing no latency during inference. Prefix-tuning (Li and Liang, 2021) trains a task specific prefix prepended to each layer of the transformer encoder and achieves comparable performance with full parameter fine-tuning on generative tasks. Similarly, prompt-tuning (Lester et al., 2021) simplifies the additional prefix to the input embeddings, and only updates the parameters corresponding to the prompts.

While the PEFT library (Mangrulkar et al., 2022) implements these algorithms at the model level, it relies on HuggingFace models and lacks a comprehensive functionality, particularly the necessary integration with model parallelism to facilitate training of extremely large models.

Parallelism Strategies Parallelism strategies refer to the methodology of utilizing multiple GPUs to execute training or inference tasks. Data parallelism involves distributing the input data to different GPUs for computation. However, each GPU stores an identical copy of the optimizer state and model weights, which limits the maximum model size that can be trained with data parallelism to that of a single GPU. To mitigate this redundancy, Rajbhandari et al. (2020) proposes a parallelism strategy in the three stages of ZeRO, evenly partitioning the optimizer states, gradients, and weights across different GPUs. Tensor parallelism also partitions the weights evenly, while it varies the approach to partition and communicate. Specifically, whereas ZeRO-3 gathers the weight matrices, tensor parallelism all reduces the intermediate computational results. Pipeline parallelism partitions the model by layers across GPUs, requiring communication only between the layers at the split points. This strategy yields the least communication overhead.

Existing toolkits, such as HuggingFace’s Trainer (Wolf et al., 2020) and LMFlow (Diao et al., 2023), choose ZeRO-3 as parallel method. ZeRO-3 is preferred because it does not impose specific requirements on the model structure, allowing direct usage of HuggingFace models. However, it exhibits lower throughput compared to the combination of TP and PP in scenarios involving large batch size pre-training or constrained communication. CoLLiE supports the hybrid application of data parallelism, tensor parallelism, and pipeline parallelism, collectively termed as 3D parallelism, with the parallel sizes adjustable via CollieConfig.
3 CoLLiE

In this section, we will introduce the implementation and features of CoLLiE. Figure 2(a) presents an overview of the Collie’s overall structure, centered around the Trainer class. The CollieConfig, CollieModel, CollieDataset, and Optimizer classes serve as inputs to the Trainer. CoLLiE also provides a set of convenient plugins for the Trainer, including Callback, Monitor, Evaluator, and Probe, enabling users to customize the training process. Depending upon the configurations and selected plugins, the Trainer performs the training process, saves model checkpoints, and records system metrics, including loss, throughput, and evaluation results.

As shown in Figure 2(b), based on PyTorch (Paszke et al., 2019) and DeepSpeed (Rasley et al., 2020), CoLLiE employs a collaborative approach using various techniques to facilitate the more efficient training of large language models. Specifically, CoLLiE integrates FlashAttention to enhance efficiency. It implements ZeRO-powered DP, TP, and PP to support 3D parallelism. Additionally, LOMO and PEFT methods are incorporated to realize memory-efficient fine-tuning approaches.

Appendix A provides a brief tour demonstrating how to use CoLLiE for training.

3.1 Collaborative Tuning Methods

3.1.1 3D Parallelism

While distributed training frameworks such as DeepSpeed and Colossal-AI (Bian et al., 2021) support 3D parallelism, models in HuggingFace can only opt for ZeRO-3 for model parallelism due to structural constraints. To fully support 3D parallelism and meet the distributed training needs under different scenarios, CoLLiE rewrites the models using Megatron-LM (Shoeybi et al., 2019) and restructures them according to DeepSpeed’s structure requirements for pipeline models. In the rewriting process, we have maintained the interface to be essentially consistent with the HuggingFace models, and have allowed the direct use of the from_pretrained method to load pre-trained models from the HuggingFace hub. This approach significantly reduces the learning curve for users.

3.1.2 Parameter-efficient Fine-tuning

The PEFT library implements state-of-the-art PEFT methods at the model level, but lacks distributed training capabilities. CoLLiE has integrated the PEFT library into CollieModel, and made necessary patches to enable distributed training.

3.1.3 Efficient Optimizers

In addition to the popular AdamW (Kingma and Ba, 2015) optimizer, several other optimizers have been proposed for the purpose of saving memory, improving optimization results, or accelerating convergence. The implementation of the optimizers in CoLLiE is decoupled from other parts, and incorporates a variety of novel optimizers including Adan, Lion, LOMO, and Sophia. The effectiveness of these optimizers in training large language models is verified and compared in Section 4.3.

3.2 Models

In addition to the above-mentioned model implementations, CoLLiE has also replaced the naïve self-attention implementation with FlashAttention. Given that FlashAttention has strict requirements regarding hardware and CUDA versions, for users without newer training equipment, we have added the ‘use_flash’ option to the CollieConfig to allow for one-click disabling of FlashAttention usage. Currently, CoLLiE has implemented a variety of language models, including but not limited to LLaMA, InternLM (Team, 2023), ChatGLM (Du et al., 2022), and MOSS (Sun et al., 2023a), with the intention to support more models in the future.

3.3 Configuration

CoLLiE offers a unified class, CollieConfig, to manage configurations including model config, parallelism strategy, DeepSpeed configuration, PEFT configuration, and training hyperparameters. Based on the contents of CollieConfig, CollieModel will automatically adjust the partitioning of model parameters and the structure of the model, and the Trainer will modify the training process. Through CollieConfig, users can conveniently combine different pre-trained language models, fine-tuning methods, and hyperparameters.

Model config refers to parameters that describe the model structure, such as hidden_size, num_attention_heads, and num_hidden_layers. The model config is fixed for pre-trained language models, and we provide a from_pretrained interface, identical to HuggingFace’s, to initialize model config. Users can also specify the model config to customize their models, intended for training from scratch without the use of pre-trained models. Below is a code...
example of downloading the model config from the HuggingFace hub, initializing the CollieConfig, and setting up to use FlashAttention.

```python
config = CollieConfig.from_pretrained('meta-llama/Llama-2-7b-hf')
config.use_flash = True
```

CollieConfig streamlines the setup for 3D parallelism as followings. CoLLiE will automatically configure the distributed environment and partition the parameters according, relieving users from managing the complexities of distributed training. The number of GPUs required for training is equal to the product of the three parallelism sizes.

```python
config.dp_size = 1
config.tp_size = 8
config.pp_size = 2
```

CoLLiE implements distributed training based on DeepSpeed, and DeepSpeed-related configurations can be set via ds_config. The configurations related to PEFT methods can also be set via peft_config. Below is an example for mixed-precision training with FP16 and LoRA.

```python
config.ds_config = {
    'fp16': {'enabled': True}
}
config.peft_config = LoraConfig(
    r=4,
    lora_alpha=32,
    target_modules=['q_proj', 'v_proj'],
    bias='none',
    task_type='CAUSAL_LM'
)
```

The training hyperparameters can also be configured through CollieConfig. Loading CollieConfig from a file is supported and we provide a convenient Command Line Interface (CLI) to generate the required configuration file.

### 3.4 Dataset

To facilitate data processing, CoLLiE provides three Dataset classes for training, evaluation of generation tasks, and evaluation of classification tasks respectively: CollieDatasetForTraining, CollieDatasetForGeneration, CollieDatasetForClassification. These three classes can either read data from a JSON file or a list of dictionaries, process it, and store the results on disk for direct reading next time.

CollieDatasetForTraining accepts two forms of input, one with the field “text”, and the other with fields “input” and “output”. The loss of tokens in the field “text” or “output” will be computed, corresponding to pre-training and instruction-tuning tasks, respectively. CollieDatasetForGeneration and CollieDatasetForClassification both inherit from the CollieDatasetForTraining class, serving as the datasets for generation tasks and classification tasks, respectively. The CollieDatasetForGeneration can accept “text” as a required field and “target” as an optional field. The model generates output based on the “text” and the “target” is used to compute metrics in Evaluator. On the other hand, CollieDatasetForClassification can accept “input”, “output”, and “target” fields. The “input” represents the question, “output” includes all possible options, and “target” indicates which
option should be chosen.

3.5 Controller

In this section, we will introduce three modularly designed classes centered around Trainer. Trainer calls the Evaluator and Server classes unidirectionally to serve the purposes of evaluation or manual probing of the model during training.

3.5.1 Trainer

Distributed training, including the initialization of the distributed environment, training loop, and the saving of model weights and checkpointing, can be complex. CoLLiE provides a Trainer to alleviate this burden on users. The Trainer wraps the relatively fixed training loop and offers multiple interfaces for users to further customize the training process. These include the train_fn function that obtains output based on a given batch of input and the loss_fn function that obtains loss based on the batch and output from train_fn. Moreover, CoLLiE offers several plugins to enrich functionality. Monitor The Monitor class tracks various metrics such as loss, learning rate, throughput, and memory usage during the training process, and records them to Tensorboard, WandB, or local CSV files. Callback The Callback class can be invoked at various callback points during the training process, allowing users to customize the training loop. CoLLiE has implemented callbacks that save model weights and training checkpoints when necessary, or load the model weights of the best evaluated results after training. These callbacks are all implemented based on the same base class. Users can inherit from this base class and override different methods to choose the callback timing and actions.

3.5.2 Evaluator

The Evaluator class is used in conjunction with the Metric class for assessing model performance. We implemented three types of Evaluator, intended for generation tasks, classification tasks, and perplexity assessment, by subclassing the Evaluator base class and overriding the eval_fn method. The return value of the eval_fn method in the Evaluator is accepted as input by the update function of the Metric class. The Metric class’s update method updates the variables necessary for calculating the metric after processing each batch from the evaluation dataset. After the evaluation dataset is fully processed, the get_metric function is employed to compute the metric. The Evaluator class can either be provided to the Trainer for assessment during the training process or it can evaluate the model independently without the dependency on the Trainer.

3.5.3 Server

The Server class offers web-based, interactive and streaming generated sequences feature, enabling users to conveniently deploy trained models for web-based use, as well as manually probe model performance during training. TheDataProvider class supplies asynchronous inference data for Server as a subprocess. When the Server is integrated into the Trainer, users can input prompts via the web interface. Once the current batch training is completed, an output will be generated based on the user’s input prompt and returned to the web interface for user’s review.

3.6 Documentation

We provide API documentation and easily understandable tutorials1 for users who are new to CoLLiE and distributed training. Comprehensive code examples2 including vocabulary expansion, instruction-tuning, and downstream tasks such as summary and translation are also available.

4 Evaluation

4.1 Memory Requirement

While Rajbhandari et al. (2020) estimates the total GPU memory required for model training as 18 times the number of model parameters in bytes, more GPU memory is consumed in reality. This is because this estimation only considers memory used by parameters, gradients, and the optimizer

1https://openlmlab-collie.readthedocs.io/zh_CN/latest
2https://github.com/OpenLMLab/collie/tree/main/examples
effectiveness of different training methods are also empirically assessed on instruction-tuning tasks.

We take HuggingFace models with ZeRO-3 as a baseline to analyse the throughput of CoLLiE during pre-training (with batch size of 1024) and fine-tuning (with batch size of 128). The corpus we used consists of the first 10,000 entries from the Pile (Gao et al., 2021). The throughput is measured by the number of tokens processed by each GPU per second, referred to as TGS.

As shown in Figure 4, on the A100 connected by NVLink, CoLLiE’s throughput significantly surpasses the baseline attributed to the integration of FlashAttention. On the RTX-3090, where communication is limited by PCIe, CoLLiE achieves substantially higher throughput by a more appropriate parallelism approach, namely TP and PP.

### 4.3 Empirical Assessment of Effectiveness

We employ various training methods on GPT-4-Alpaca (Peng et al., 2023) for the LLaMA-65B model and evaluate the factual knowledge, reasoning abilities, code capabilities, and instruction-following abilities using MMLU (Hendrycks et al., 2021), BBH (Suzgun et al., 2023), GSM8K (Cobbe et al., 2021), HumanEval (Chen et al., 2021), and AlpacaFarm (Dubois et al., 2023). The hyperparameters and templates for training and evaluating can be found in Appendix B.3 and Appendix C, respectively.

The results in Table 1 demonstrate that while the vanilla LLaMA-65B already exhibits substantial capabilities, it struggles to effectively follow instructions from actual users. The performance of the models significantly improves on average after instruction-tuning. Training methods such as LoRA, LOMO, and AdamW significantly enhance the model’s ability to follow instructions without compromising its other performance.

### 5 Conclusion

We have introduced CoLLiE, a library for collaboratively training large language models in an efficient way. CoLLiE offers efficient models with FlashAttention and structurally supportive for 3D parallelism. Moreover, CoLLiE provides a comprehensive and customizable Trainer to assist users throughout the training process, supporting various training methods. We have tested the relationship between the GPU memory requirements and model parameter sizes as a reference for users. In terms of throughput, CoLLiE is significantly more efficient than HuggingFace’s parallel solutions. The effectiveness of different training methods are also empirically assessed on instruction-tuning tasks.

<table>
<thead>
<tr>
<th>Method</th>
<th>BBH</th>
<th>GSM</th>
<th>HumanEval</th>
<th>AlpacaFarm</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>62.4</td>
<td>56.9</td>
<td>53.9</td>
<td>20.7</td>
<td>4.7</td>
</tr>
<tr>
<td>LoRA</td>
<td>62.7</td>
<td><strong>60.5</strong></td>
<td><strong>32.9</strong></td>
<td>69.6</td>
<td><strong>56.9</strong></td>
</tr>
<tr>
<td>LOMO</td>
<td>62.1</td>
<td>56.9</td>
<td>57.6</td>
<td>28.1</td>
<td>65.2</td>
</tr>
<tr>
<td>Lion</td>
<td>58.2</td>
<td>52.6</td>
<td>41.3</td>
<td>25.0</td>
<td>66.2</td>
</tr>
<tr>
<td>Adam</td>
<td>57.3</td>
<td>51.9</td>
<td>37.3</td>
<td>21.3</td>
<td>62.5</td>
</tr>
<tr>
<td>AdamW</td>
<td><strong>63.0</strong></td>
<td>58.0</td>
<td>55.3</td>
<td>28.1</td>
<td><strong>73.1</strong></td>
</tr>
</tbody>
</table>

Table 1: Comparison of different training methods on GPT4-Alpaca. Instruction-tuning significantly enhances the instruction-following ability of vanilla LLaMA-65B.
Limitations

We discuss the limitations of this paper from the following two aspects:

1) Although we profile the memory usage under real training conditions in this paper, a more fine-grained memory allocation situation is not provided. In the future, we plan to develop a fine-grained memory monitor to assist users in training.

2) Due to resource and time constraints, this paper only presents the instruction-tuning results of LLaMA-65B with different training methods. This restricts users from comparing the performance of models of different sizes. We will provide performances of more models under various training methods and continuously update them on our Github repository for user reference. Furthermore, while CoLLiE has implemented the Sophia optimizer to enhance pre-training efficiency, we have not conducted extensive experiments under costly pre-training tasks.

Acknowledgements

This work was supported by the National Key Research and Development Program of China (No.2022ZD0160102) and National Natural Science Foundation of China (No.62022027).

References


Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In KDD ’20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020, pages 3505–3506. ACM.

A Code Example

Listing 1 presents the simplest code example for training with CoLLiE.

B Hyperparameters

<table>
<thead>
<tr>
<th>Training Methods</th>
<th>LR</th>
<th>Batch Size</th>
<th>Weight Decay</th>
<th>Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoRA</td>
<td>3e-4</td>
<td>128</td>
<td>1e-2</td>
<td>3</td>
</tr>
<tr>
<td>LOMO</td>
<td>1e-2</td>
<td>16</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>Lion</td>
<td>3e-6</td>
<td>128</td>
<td>3e-2</td>
<td>3</td>
</tr>
<tr>
<td>Adan</td>
<td>5e-5</td>
<td>128</td>
<td>2e-2</td>
<td>3</td>
</tr>
<tr>
<td>Adam</td>
<td>1e-5</td>
<td>128</td>
<td>1e-2</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2: Hyperparameters for training.

B.1 Memory Requirements

We choose the combination of Tensor Parallelism (TP) and Pipeline Parallelism (PP) as our parallelism strategy. The batch size is set to 2048, and the gradient accumulation steps are set to 2. It’s worth noting that increasing the value of the gradient accumulation steps would not significantly increase the memory usage.

B.2 Throughput

In our throughput tests, we consistently employ Adam as the optimizer. We utilize the default settings of DeepSpeed for ZeRO-3 and strive to maximize the micro batch size to enhance throughput. For Tensor Parallelism/Pipeline Parallelism (TP/PP), we ensure that the gradient accumulation steps are more than four times the number of pipeline stages to minimize the bubble. The specific configurations are illustrated in Table 4.

B.3 Instruction-tuning

As shown in Table 2, we have adopted the learning rates and batch sizes from the Tulu (Wang et al., 2023) and Alpaca-LoRA projects\(^3\) for AdamW and LoRA. To achieve better performance for LoRA, we have replaced all modules with LoRA layers, not just the q-v module. For Lion and Adan, we have used the learning rates recommended in the paper. Specifically, the learning rate for Lion is 3-10 times smaller than that of AdamW, with the weight decay correspondingly 3-10 times larger. The learning rate for the Adan optimizer is 5-10 times larger than that of AdamW, with a weight decay of 0.02. For the LOMO optimizer, which is similar to SGD, we have utilized a larger learning rate and a smaller batch size.

C Templates

C.1 Alpaca

We follow the template provided by the Alpaca repository\(^4\) for training, as shown in Table 3.

C.2 Evaluation

We modify the evaluation template based on the template used during training, as shown in Table 5. The template used for evaluation on AlpacaFarm is identical to that of training on Alpaca.

---

\(^3\)https://github.com/tloen/alpaca-lora

\(^4\)https://github.com/tatsu-lab/stanford_alpaca
Listing 1: An example for training with CoLLiE.

```python
import torch
from collie.config import CollieConfig
from collie.models import LlamaForCausalLM
from collie.controller import Trainer

model_name_or_path = 'meta-llama/Llama-2-7b-hf'
config = CollieConfig.from_pretrained(model_name_or_path)
config.pp_size = 8
model = LlamaForCausalLM.from_pretrained(model_name_or_path, config=config)
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
train_dataset = [
    {
        'text': 'Collie is a package for training large language models.'
    }
    for _ in range(100)
]
trainer = Trainer(model=model, optimizer=optimizer, config=config, train_dataset=train_dataset)
trainer.train()
```

Table 4: Hyperparameters for testing throughput. We report the number of model parameters (Model Params), device (Device), mode (Mode) and number of GPU (#GPU). We also report the corresponding batch size (Batch Size) and GAS (Gradient Accumulation Steps) for HuggingFace with ZeRO-3, CoLLiE with ZeRO-3 and CoLLiE with TP/PP.
Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
The following is a multiple choice question (with answers) about abstract algebra. You need to answer the question by selecting the correct option.

### Input:
Find all c in \( \mathbb{Z}_3 \) such that \( \mathbb{Z}_3[x]/(x^2 + c) \) is a field.
A. 0
B. 1
C. 2
D. 3

### Response: B

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
The following is multiple choice question (with answers) about abstract algebra. You need to answer the question by selecting the correct option.

### Input:
Statement 1 | If \( aH \) is an element of a factor group, then \( |aH| \) divides \( |a| \). Statement 2 | If \( H \) and \( K \) are subgroups of \( G \) then \( HK \) is a subgroup of \( G \).
A. True, True
B. False, False
C. True, False
D. False, True

### Response: B

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
The following is multiple choice question (with answers) about abstract algebra. You need to answer the question by selecting the correct option.

### Input:
Statement 1 | Every element of a group generates a cyclic subgroup of the group. Statement 2 | The symmetric group \( S_{10} \) has 10 elements.
A. True, True
B. False, False
C. True, False
D. False, True

### Response: C

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
The following is multiple choice question (with answers) about abstract algebra. You need to answer the question by selecting the correct option.
### Input:
Statement 1 | Every function from a finite set onto itself must be one to one. Statement 2 | Every subgroup of an abelian group is abelian.
A. True, True
B. False, False
C. True, False
D. False, True

### Response: A

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
The following is multiple choice question (with answers) about abstract algebra. You need to answer the question by selecting the correct option.

### Input:
Find the characteristic of the ring 2Z.
A. 0
B. 3
C. 12
D. 30

### Response: A

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
The following is multiple choice question (with answers) about abstract algebra. You need to answer the question by selecting the correct option.

### Input:
Input

### Response:

BBH

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
Evaluate the result of a random Boolean expression.

### Input:
not (( not not True ) ) is

### Response: Let’s think step by step.
Remember that (i) expressions inside brackets are always evaluated first and that (ii) the order of operations from highest priority to lowest priority is “not”, “and”, “or”, respectively.
We first simplify this expression “Z” as follows: “Z = not ( ( not not True ) ) = not ( ( A ) )” where “A = not not True”.
Let’s evaluate A: A = not not True = not (not True) = not False = True.
Plugging in A, we get: Z = not ( ( A ) ) = not ( ( True ) ) = not True = False. So the answer is False.

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.
Evaluate the result of a random Boolean expression.

True and False and not True and True is

Let’s think step by step.
Remember that (i) expressions inside brackets are always evaluated first and that (ii) the order of operations from highest priority to lowest priority is “not”, “and”, “or”, respectively.
We first simplify this expression “Z” as follows: “Z = True and False and not True and True = A and B” where “A = True and False” and “B = not True and True”.
Let’s evaluate A: A = True and False = False.
Let’s evaluate B: B = not True and True = not (True and True) = not (True) = False.
Plugging in A and B, we get: Z = A and B = False and False = False. So the answer is False.

Evaluate the result of a random Boolean expression.

not not ( not ( False ) ) is

Let’s think step by step.
Remember that (i) expressions inside brackets are always evaluated first and that (ii) the order of operations from highest priority to lowest priority is “not”, “and”, “or”, respectively.
We first simplify this expression ”Z” as follows: “Z = not not ( not ( False ) ) = not not ( A )” where “A = not ( False )”.
Let’s evaluate A: A = not ( False ) = not False = True.
Plugging in A, we get: Z = not not ( A ) = not not (True) = not not False = True. So the answer is True.
Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

Given a problem scenario with numerical data, perform the necessary calculations and provide a detailed step-by-step solution, ending the response with ‘The answer is’.

Angelo and Melanie want to plan how many hours over the next week they should study together for their test next week. They have 2 chapters of their textbook to study and 4 worksheets to memorize. They figure out that they should dedicate 3 hours to each chapter of their textbook and 1.5 hours for each worksheet. If they plan to study no more than 4 hours each day, how many days should they plan to study total over the next week if they take a 10-minute break every hour, include 3 10-minute snack breaks each day, and 30 minutes for lunch each day?
### Response: Let’s think step by step. Angelo and Melanie think they should dedicate 3 hours to each of the 2 chapters, 3 hours x 2 chapters = 6 hours total. For the worksheets they plan to dedicate 1.5 hours for each worksheet, 1.5 hours x 4 worksheets = 6 hours total. Angelo and Melanie need to start with planning 12 hours to study, at 4 hours a day, 12 / 4 = 3 days. However, they need to include time for breaks and lunch. Every hour they want to include a 10-minute break, so 12 total hours x 10 minutes = 120 extra minutes for breaks. They also want to include 3 10-minute snack breaks, 3 x 10 minutes = 30 minutes. And they want to include 30 minutes for lunch each day, so 120 minutes for breaks + 30 minutes for snack breaks + 30 minutes for lunch = 180 minutes, or 180 / 60 minutes per hour = 3 extra hours. So Angelo and Melanie want to plan 12 hours to study + 3 hours of breaks = 15 hours total. They want to study no more than 4 hours each day, 15 hours / 4 hours each day = 3.75 They will need to plan to study 4 days to allow for all the time they need. The answer is 4

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
Given a problem scenario with numerical data, perform the necessary calculations and provide a detailed step-by-step solution, ending the response with 'The answer is'.

### Input:
Mark’s basketball team scores 25 2 pointers, 8 3 pointers and 10 free throws. Their opponents score double the 2 pointers but half the 3 pointers and free throws. What’s the total number of points scored by both teams added together?

### Response: Let’s think step by step. Mark’s team scores 25 2 pointers, meaning they scored 25*2 = 50 points in 2 pointers. His team also scores 6 3 pointers, meaning they scored 8*3 = 24 points in 3 pointers They scored 10 free throws, and free throws count as one point so they scored 10*1 = 10 points in free throws. All together his team scored 50+24+10= 84 points Mark’s opponents scored double his team’s number of 2 pointers, meaning they scored 50*2=100 points in 2 pointers. His team also scored 6+3=9 points in 3 pointers They scored 10/2=5 points in free throws. All together Mark’s opponents scored 100+9+5=114 points The total score for the game is both team’s scores added together, so it is 84+114=201 points The answer is 201

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
Given a problem scenario with numerical data, perform the necessary calculations and provide a detailed step-by-step solution, ending the response with 'The answer is'.

### Input:
Bella has two times as many marbles as frisbees. She also has 20 more frisbees than deck cards. If she buys 2/5 times more of each item, what would be the total number of the items she will have if she currently has 60 marbles?

### Response: Let’s think step by step. When Bella buys 2/5 times more marbles, she’ll have increased the number of marbles by 2/5*60 = 24 The total number of marbles she’ll have is 60+24 = 84 If Bella currently has 60 marbles, and she has two times as many marbles as frisbees, she has 60/2 = 30 frisbees. If Bella buys 2/5 times more frisbees, she’ll have 2/5*30 = 12 more frisbees. The total number of frisbees she’ll have will increase to 30+12 = 42 Bella also has 20 more frisbees than deck cards, meaning she has 30-20 = 10 deck cards If she buys 2/5 times more deck cards, she’ll have 2/5*10 = 4 more deck cards. The total number of deck cards she’ll have is 10+4 = 14 Together, Bella will have a total of 14+42+84 = 140 items The answer is 140

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
Given a problem scenario with numerical data, perform the necessary calculations and provide a detailed step-by-step solution, ending the response with 'The answer is'.

### Input:
A group of 4 fruit baskets contains 9 apples, 15 oranges, and 14 bananas in the first three baskets and 2 less of each fruit in the fourth basket. How many fruits are there?
Let's think step by step. For the first three baskets, the number of apples and oranges in one basket is 9+15=24. In total, together with bananas, the number of fruits in one basket is 24+14=38 for the first three baskets. Since there are three baskets each having 38 fruits, there are 3*38=114 fruits in the first three baskets. The number of apples in the fourth basket is 9-2=7. There are also 15-2=13 oranges in the fourth basket. The combined number of oranges and apples in the fourth basket is 13+7=20. The fourth basket also contains 14-2=12 bananas. In total, the fourth basket has 20+12=32 fruits. The four baskets together have 32+114=146 fruits. The answer is 146.

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

### Instruction:
Given a problem scenario with numerical data, perform the necessary calculations and provide a detailed step-by-step solution, ending the response with 'The answer is'.

### Input:
{question}

### Response: Let's think step by step.

```python
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:
    '''Check if in given list of numbers, are any two numbers closer to each other than given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5) False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3) True
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5) False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3) True
    '''
    return any(abs(i - j) < threshold for i in numbers for j in numbers if i != j)
```

Table 5: The templates used for evaluation.