

```
%%capture
%pip install unsloth vllm

from unsloth import FastLanguageModel
import torch
max_seq_length = 1300 # Can increase for longer reasoning traces
lora_rank = 8 # Larger rank = smarter, but slower

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/phi-4-unsloth-bnb-4bit",
    max_seq_length = max_seq_length,
    load_in_4bit = True, # False for LoRA 16bit
    fast_inference = True, # Enable vLLM fast inference
    full_finetuning = False,
    max_lora_rank = lora_rank,
    gpu_memory_utilization = 0.7, # Reduce if out of memory
    token="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"
)

model = FastLanguageModel.get_peft_model(
    model,
    r = lora_rank, # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules = [
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj",
    ],
    lora_dropout = 0, #For smooth and fast fine-tuning but to avoid overfitting can be set to 0.1
    lora_alpha = lora_rank,
    use_gradient_checkpointing = "unsloth", # Enable long context finetuning
    random_state = 3407,
)
```

```

🔗 🐍 Unsloth: Will patch your computer to enable 2x faster free finetuning.
Unsloth: Failed to patch Gemma3ForConditionalGeneration.
🐍 Unsloth Zoo will now patch everything to make training faster!
INFO 04-21 05:02:26 [__init__.py:239] Automatically detected platform cuda.
Unsloth: Switching from Unsloth dynamic quant to normal quant since
we do not yet support fast inference for unsloth/phi-4-unsloth-bnb-4bit
==(((====))== Unsloth 2025.3.19: Fast Llama patching. Transformers: 4.51.3. vLLM: 0.8.4.
  \ \ / \ Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform: Linux.
0^0/ \_/ \ Torch: 2.6.0+cu124. CUDA: 7.5. CUDA Toolkit: 12.4. Triton: 3.2.0
 \ \ / \ Bfloat16 = FALSE. FA [Xformers = 0.0.29.post2. FA2 = False]
"-_____" Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!

config.json: 100% 1.40k/1.40k [00:00<00:00, 126kB/s]

Unsloth: vLLM loading unsloth/phi-4-bnb-4bit with actual GPU utilization = 69.34%
Unsloth: Your GPU has CUDA compute capability 7.5 with VRAM = 14.74 GB.
Unsloth: Using conservativeness = 1.0. Chunked prefill tokens = 1300. Num Sequences = 128.
Unsloth: vLLM's KV Cache can use up to 0.31 GB. Also swap space = 2 GB.
WARNING 04-21 05:02:43 [config.py:2836] Casting torch.bfloat16 to torch.float16.
INFO 04-21 05:02:57 [config.py:689] This model supports multiple tasks: {'generate', 'score', 'embed', 'reward', 'clas
WARNING 04-21 05:02:57 [arg_utils.py:1731] Compute Capability < 8.0 is not supported by the V1 Engine. Falling back to
Unsloth: vLLM Bitsandbytes config using kwargs = {'load_in_8bit': False, 'load_in_4bit': True, 'bnb_4bit_compute_dtype
INFO 04-21 05:02:57 [llm_engine.py:243] Initializing a V0 LLM engine (v0.8.4) with config: model='unsloth/phi-4-bnb-4b

tokenizer_config.json: 100% 18.0k/18.0k [00:00<00:00, 1.46MB/s]

vocab.json: 100% 1.61M/1.61M [00:00<00:00, 2.43MB/s]

merges.txt: 100% 917k/917k [00:00<00:00, 9.57MB/s]

tokenizer.json: 100% 7.15M/7.15M [00:02<00:00, 2.53MB/s]

special_tokens_map.json: 100% 570/570 [00:00<00:00, 54.2kB/s]

generation_config.json: 100% 170/170 [00:00<00:00, 17.1kB/s]
INFO 04-21 05:03:05 [cuda.py:240] Cannot use FlashAttention-2 backend for Volta and Turing GPUs.
INFO 04-21 05:03:05 [cuda.py:289] Using XFormers backend.
INFO 04-21 05:03:06 [parallel_state.py:959] rank 0 in world size 1 is assigned as DP rank 0, PP rank 0, TP rank 0
INFO 04-21 05:03:06 [model_runner.py:1110] Starting to load model unsloth/phi-4-bnb-4bit...
INFO 04-21 05:03:06 [loader.py:1166] Loading weights with BitsAndBytes quantization. May take a while ...
INFO 04-21 05:03:07 [weight_utils.py:265] Using model weights format ['*.safetensors']

model-00001-of-00002.safetensors: 100% 4.98G/4.98G [00:35<00:00, 122MB/s]

model-00002-of-00002.safetensors: 100% 4.11G/4.11G [00:36<00:00, 94.2MB/s]
INFO 04-21 05:04:19 [weight_utils.py:281] Time spent downloading weights for unsloth/phi-4-bnb-4bit: 72.482173 seconds
model.safetensors.index.json: 100% 165k/165k [00:00<00:00, 782kB/s]

Loading safetensors checkpoint shards: 100% Completed | 2/2 [00:38<00:00, 19.24s/it]

Loading safetensors checkpoint shards: 100% Completed | 2/2 [00:38<00:00, 18.76s/it]
INFO 04-21 05:05:37 [punica_selector.py:18] Using PunicaWrapperGPU.
INFO 04-21 05:05:38 [model_runner.py:1146] Model loading took 8.5612 GiB and 151.829778 seconds
INFO 04-21 05:05:54 [worker.py:267] Memory profiling takes 14.99 seconds
INFO 04-21 05:05:54 [worker.py:267] the current vLLM instance can use total_gpu_memory (14.74GiB) x gpu_memory_utiliza
INFO 04-21 05:05:54 [worker.py:267] model weights take 8.56GiB; non_torch_memory takes 0.03GiB; PyTorch activation pea
INFO 04-21 05:05:55 [executor_base.py:112] # cuda blocks: 378, # CPU blocks: 655
INFO 04-21 05:05:55 [executor_base.py:117] Maximum concurrency for 1300 tokens per request: 4.65x
INFO 04-21 05:05:57 [model_runner.py:1456] Capturing cudagraphs for decoding. This may lead to unexpected consequences
Capturing CUDA graph shapes: 100% 19/19 [01:21<00:00, 4.67s/it]
INFO 04-21 05:07:18 [model_runner.py:1598] Graph capturing finished in 82 secs, took 0.62 GiB
INFO 04-21 05:07:18 [llm_engine.py:449] init engine (profile, create kv cache, warmup model) took 100.07 seconds
tokenizer_config.json: 100% 18.0k/18.0k [00:00<00:00, 1.68MB/s]

vocab.json: 100% 1.61M/1.61M [00:00<00:00, 19.2MB/s]

merges.txt: 100% 917k/917k [00:00<00:00, 2.12MB/s]

special_tokens_map.json: 100% 570/570 [00:00<00:00, 50.0kB/s]

tokenizer.json: 100% 7.15M/7.15M [00:01<00:00, 4.80MB/s]
Unsloth 2025.3.19 patched 40 layers with 40 QKV layers, 40 O layers and 40 MLP layers.

```

```

from google.colab import drive
drive.mount('/content/drive')

```

```

🔗 Mounted at /content/drive

```

```

import os
import re
import pandas as pd
#Preparing the dataset
data_dir = '/content/drive/MyDrive/Dataset'
data_records = []

def normalize_citations_consistent(text):

```

```

"""
Normalize citations by converting different citation formats
into a standardized placeholder "[Citation]".
Detects numeric citations like "[12]" or "[12, 34]" and
author-year citations like "(Smith et al., 2020)" or "Smith et al. (2020)".
"""

# Replace numeric citations in square brackets.
text = re.sub(r'\[\s*\d+(?:\s*,\s*\d+)*\s*\]', '[Citation]', text)
# Replace citations in the form "(Smith et al., 2020)" or "(Smith, 2020)".
text = re.sub(r'\([A-Z][a-zA-Z]*(?: et al\.)?\s*\d{4}\)', '[Citation]', text)
# Also handle cases like "Smith et al. (2020)" without outer parentheses.
text = re.sub(r'[A-Z][a-zA-Z]*(?: et al\.)?\s*\d{4}\s*\)', '[Citation]', text)
# Normalize excessive whitespace.
text = re.sub(r'\s+', ' ', text).strip()
return text

def extract_title_and_abstract(text):
    """
    Extracts the title and abstract from text using 'Title:' and 'Abstract:' keywords.
    Falls back to treating the whole text as abstract if title is not found.

    Returns:
        (title, abstract): Tuple of extracted title and abstract
    """
    title = ""
    abstract = text.strip()

    # Normalize line endings
    text = text.replace('\r\n', '\n').replace('\r', '\n')

    # Extract title (case-insensitive, from first "Title:")
    title_match = re.search(r'\bTitle:\s*(.*)', text, re.IGNORECASE)
    if title_match:
        title = title_match.group(1).strip()
        # Remove title line from text
        text = re.sub(r'\bTitle:\s*.*\n?', '', text, flags=re.IGNORECASE, count=1).strip()

    # Extract abstract (case-insensitive, from first "Abstract:")
    abstract_match = re.search(r'\bAbstract:\s*(.*)', text, re.IGNORECASE | re.DOTALL)
    if abstract_match:
        abstract = abstract_match.group(1).strip()

    return title, abstract

# Iterate through each subfolder (e.g. folders named "cancer" or "non-cancer").
for folder in os.listdir(data_dir):
    folder_path = os.path.join(data_dir, folder)
    if os.path.isdir(folder_path):
        # Determine the label based on folder name.
        label = "Noncancer" if "non-cancer" in folder.lower() else "Cancer"
        for file in os.listdir(folder_path):
            if file.endswith('.txt'):
                file_path = os.path.join(folder_path, file)
                with open(file_path, 'r', encoding='utf-8') as f:
                    text = f.read().strip()
                # Normalize citations.
                text = normalize_citations_consistent(text)
                # Extract title and abstract.
                title, abstract_text = extract_title_and_abstract(text)
                abstract_id = os.path.splitext(file)[0]
                data_records.append({
                    "question": re.sub(r"<ID:[^>]+>\s*", "", abstract_text),
                    "answer": label
                })

# Create a Pandas DataFrame with the loaded data:
df = pd.DataFrame(data_records)
df.head()

```



	question	answer	
0	Title: DNA/MVA Vaccination of HIV-1 Infected P...	Noncancer	
1	Title: Metabolic reprogramming is required for...	Noncancer	
2	Title: Splenic cyst and its management in a 21...	Noncancer	
3	Title: Methods to Study DNA End Resection I: R...	Noncancer	
4	Title: Ringed telangiectasias: an unusual pres...	Noncancer	

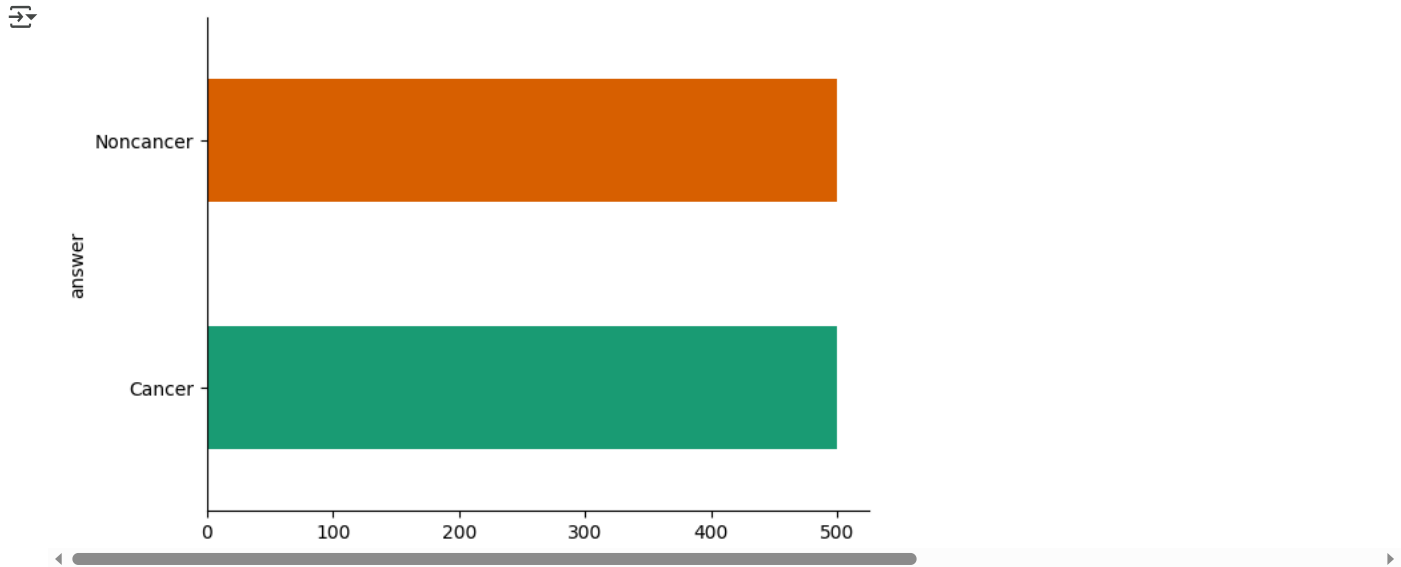
Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

```

from matplotlib import pyplot as plt
import seaborn as sns
df.groupby('answer').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
plt.gca().spines[['top', 'right']].set_visible(False)

```



```

system_prompt = \
f"""
You are given a research abstract.
Think about the research abstract, understand it.
Then, Classify the research abstract into either "Cancer" or "Noncancer" category.

```

Respond in the following format:

```

<reasoning>
...
</reasoning>
<answer>
...
</answer>
"""

```

```
system_prompt
```

```

'\nYou are given a research abstract.\nThink about the research abstract, understand it.\nThen, Classify the research abstract into either "Cancer" or "Noncancer" category.\n\nRespond in the following format:\n<reasoning>\n...\n</reasoning>

```

```

XML_COT_FORMAT = """\
<reasoning>
{reasoning}
</reasoning>
<answer>
{answer}
</answer>
"""

```

```


from datasets import Dataset
df = df.sample(frac=1, random_state=85)
# Split the DataFrame
train_size = 0.8
eval_size = 0.1

# Calculate sizes
train_end = int(train_size * len(df))
eval_end = train_end + int(eval_size * len(df))

# Split the data
X_train = df[:train_end]
X_eval = df[train_end:eval_end]
X_test = df[eval_end:]

X_train.answer.value_counts()

```




	count
answer	
Noncancer	410
Cancer	390

dtypes: int64


```
train_data = Dataset.from_pandas(X_train)
eval_data = Dataset.from_pandas(X_eval)
test_data = Dataset.from_pandas(X_test)
```

```
train_data = train_data.map(lambda x: {
    "prompt" : [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": x["question"]},
    ],
    "answer": (x["answer"]),
})
train_data[0]
```

 Map: 100% 800/800 [00:00<00:00, 7418.48 examples/s]

```
{'question': 'Title: Immunotherapy in colorectal cancer: rationale, challenges and potential. Abstract: Following initial successes in melanoma treatment, immunotherapy has rapidly become established as a major treatment modality for multiple types of solid cancers, including a subset of colorectal cancers (CRCs). Two programmed cell death 1 (PD1)-blocking antibodies, pembrolizumab and nivolumab, have shown efficacy in patients with metastatic CRC that is mismatch-repair-deficient and microsatellite instability-high (dMMR-MSI-H), and have been granted accelerated FDA approval. In contrast to most other treatments for metastatic cancer, immunotherapy achieves long-term durable remission in a subset of patients, highlighting the tremendous promise of immunotherapy in treating dMMR-MSI-H metastatic CRC. Here, we review the clinical development of immune checkpoint inhibition in CRC leading to regulatory approvals for the treatment of dMMR-MSI-H CRC. We focus on new advances in expanding the efficacy of immunotherapy to early-stage CRC and CRC that is mismatch-repair-proficient and has low microsatellite instability (pMMR-MSI-L) and discuss emerging approaches for targeting the immune microenvironment, which might complement immune checkpoint inhibition.',
'answer': 'Cancer',
'__index_level_0__': 866,
'prompt': [{'content': '\nYou are given a research abstract.\nThink about the research abstract, understand it.\nThen, Classify the research abstract into either "Cancer" or "Noncancer" category.\n\nRespond in the following format:\n<reasoning>\n...\n</reasoning>\n<answer>\n...\n</answer>\n',
'role': 'system'},
{'content': 'Title: Immunotherapy in colorectal cancer: rationale, challenges and potential. Abstract: Following initial successes in melanoma treatment, immunotherapy has rapidly become established as a major treatment modality for multiple types of solid cancers, including a subset of colorectal cancers (CRCs). Two programmed cell death 1 (PD1)-
```

```
eval_data = eval_data.map(lambda x: {
    "prompt" : [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": x["question"]},
    ],
    "answer": (x["answer"]),
})
eval_data[0]
```

 Map: 100% 100/100 [00:00<00:00, 3378.66 examples/s]

```
{'question': 'Title: Defining the role of common variation in the genomic and biological architecture of adult human height. Abstract: Using genome-wide data from 253,288 individuals, we identified 697 variants at genome-wide significance that together explained one-fifth of the heritability for adult height. By testing different numbers of variants in independent studies, we show that the most strongly associated ~2,000, ~3,700 and ~9,500 SNPs explained ~21%, ~24% and ~29% of phenotypic variance. Furthermore, all common variants together captured 60% of heritability. The 697 variants clustered in 423 loci were enriched for genes, pathways and tissue types known to be involved in growth and together implicated genes and pathways not highlighted in earlier efforts, such as signaling by fibroblast growth factors, WNT/beta-catenin and chondroitin sulfate-related genes. We identified several genes and pathways not previously connected with human skeletal growth, including mTOR, osteoglycin and binding of hyaluronic acid. Our results indicate a genetic architecture for human height that is characterized by a very large but finite number (thousands) of causal variants.',
'answer': 'Noncancer',
'__index_level_0__': 98,
'prompt': [{'content': '\nYou are given a research abstract.\nThink about the research abstract, understand it.\nThen, Classify the research abstract into either "Cancer" or "Noncancer" category.\n\nRespond in the following format:\n<reasoning>\n...\n</reasoning>\n<answer>\n...\n</answer>\n',
'role': 'system'},
{'content': 'Title: Defining the role of common variation in the genomic and biological architecture of adult human
```

```
test_data = test_data.map(lambda x: {
    "prompt" : [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": x["question"]},
    ],
})
test_data[0]
```



Map: 100%

100/100 [00:00<00:00, 3165.92 examples/s]

```
{'question': 'Title: Analysis of LPL gene expression in patients with chronic lymphocytic leukemia. Abstract: AIM: The IGHV mutational status is one of the most important markers for chronic lymphocytic leukemia (CLL) prognostication. Lipoprotein lipase (LPL) gene expression was found to correlate with IGHV status and was suggested as its surrogate marker. Recent data reported that LPL expression might be influenced by pivotal signalling pathways in CLL. This study aimed to assess LPL gene expression in relation to key immunogenetic and molecular markers of CLL, including IGHV mutational status, B-cell receptor (BCR) stereotypy, TP53, NOTCH1, and SF3B1 gene mutations. Materials and Methods: Expression of LPL mRNA was measured in peripheral blood mononuclear cells of 73 CLL patients by real-time quantitative reverse transcription polymerase chain reaction (RT-qPCR). IGHV, NOTCH1, TP53, and SF3B1 gene mutation analysis was performed by PCR amplification and direct sequencing. RESULTS: 44 of 73 (60%) CLL cases were categorized as LPL-positive based on the cut-off value established by ROC (receiver operating characteristic) curve analysis. LPL expression was significantly associated with IGHV mutation status ( $r = 0.684$ ;  $p < 0.0001$ ) and tended to correlate with presence of NOTCH1 gene mutations ( $p = 0.113$ ). BCR stereotyped cases showed higher LPL expression values in comparison to unstereotyped cases in the LPL-positive group of patients ( $p = 0.041$ ). LPL expression was associated with a shorter overall survival in the entire SLL group (median 107 vs 143,  $p = 0.048$ ) as well as in Binet A patients, albeit with borderline significance (median 139 vs not reached,  $p = 0.086$ ). CONCLUSION: LPL expression was found to be closely correlated with IGHV gene mutational status and overall survival, proving LPL as prognostic marker in CLL. Our results also indicate a possible relationship between aberrant expression of LPL and BCR- and NOTCH1-dependent signalling pathways.',
'answer': 'Cancer',
'__index_level_0__': 506,
'prompt': [{'content': '\nYou are given a research abstract.\nThink about the research abstract, understand it.\nThen, Classify the research abstract into either "Cancer" or "Noncancer" category.\n\nRespond in the following format:\n<reasoning>\n...\n</reasoning>\n<answer>\n...\n</answer>\n',
'role': 'system'}],
'content': 'Title: Analysis of LPL gene expression in patients with chronic lymphocytic leukemia. Abstract: AIM: The IGHV mutational status is one of the most important markers for chronic lymphocytic leukemia (CLL) prognostication.'}
```

```
import json
import re
import numpy as np

# Define a regex pattern to verify the JSON format exactly.
match_labels = re.compile(r'\b(Non[-\s]?Cancer|Cancer)\b', flags=re.IGNORECASE)

def extract_xml_answer(text: str) -> str:
    answer = text.split("<answer>")[-1]
    answer = answer.split("</answer>")[0]
    return answer.strip()

def strict_format_reward_func(completions, **kwargs) -> list[float]:
    """Reward function that checks if the completion has a specific format."""
    pattern = r"^(?<reasoning>\n.*?\n</reasoning>\n<answer>\n.*?\n</answer>\n$"
    responses = [completion[0]["content"] for completion in completions]
    matches = [re.match(pattern, r) for r in responses]
    return [0.5 if match else 0.0 for match in matches]

def soft_format_reward_func(completions, **kwargs) -> list[float]:
    """Reward function that checks if the completion has a specific format."""
    pattern = r"<reasoning>.*?</reasoning>\s*<answer>.*?</answer>"
    responses = [completion[0]["content"] for completion in completions]
    matches = [re.match(pattern, r) for r in responses]
    return [0.5 if match else 0.0 for match in matches]

def count_xml(text) -> float:
    count = 0.0
    if text.count("<reasoning>\n") == 1:
        count += 0.125
    if text.count("\n</reasoning>\n") == 1:
        count += 0.125
    if text.count("\n<answer>\n") == 1:
        count += 0.125
    count -= len(text.split("\n</answer>\n")[-1])*0.001
    if text.count("\n</answer>") == 1:
        count += 0.125
    count -= (len(text.split("\n</answer>")[-1]) - 1)*0.001
    return count

def normalize_label(text: str) -> str:
    matches = match_labels.findall(text)
    if matches:
        label = matches[-1] # last match
        return label.replace("-", "").replace(" ", "").capitalize()
    return "unknown"

def correctness_reward_func(prompts, completions, answer, **kwargs) -> list[float]:
    responses = [completion[0]['content'] for completion in completions]
    q = prompts[0][-1]['content']
    extracted_responses = [normalize_label(r) for r in responses]
    expected_answer = normalize_label(answer[0])
    return [2.0 if r == expected_answer else 0.0 for r in extracted_responses]

def xmlcount_reward_func(completions, **kwargs) -> list[float]:
```

```

contents = [completion[0]["content"] for completion in completions]
return [count_xml(c) for c in contents]

# Also sometimes it might not be 1 label as the answer, but like a sentence for example "The solution is Cancer" -> we extract
# Case-insensitive pattern to match "Cancer" or "Non-Cancer"

# Examples
examples = [
    "The solution is Cancer",
    "The solution is Non-Cancer",
    "<SOLUTION> Non-Cancer </SOLUTION>",
    "<SOLUTION> Cancer </SOLUTION>",
    "<answer> Non-Cancer </answer>",
    "<answer> Cancer </answer>",
    "Likely non cancer based on findings",
]

for ex in examples:
    print(normalize_label(ex))

Cancer
Noncancer
Noncancer
Cancer
Noncancer
Cancer
Noncancer

def drop_index_key(example):
    if '__index_level_0__' in example:
        del example['__index_level_0__']
    return example

train_data = train_data.map(drop_index_key)
eval_data = eval_data.map(drop_index_key)
test_data = eval_data.map(drop_index_key)

Map: 100%                               800/800 [00:00<00:00, 11971.24 examples/s]
Map: 100%                               100/100 [00:00<00:00, 3071.15 examples/s]
Map: 100%                               100/100 [00:00<00:00, 4229.07 examples/s]

max(train_data.map(
    lambda x: {"tokens" : tokenizer.apply_chat_template(x["prompt"], add_generation_prompt = True, tokenize = True)},
    batched = True,
).map(lambda x: {"length" : len(x["tokens"])}))["length"])

Map: 100%                               800/800 [00:01<00:00, 673.93 examples/s]
Map: 100%                               800/800 [00:00<00:00, 2249.64 examples/s]
1637

max(eval_data.map(
    lambda x: {"tokens" : tokenizer.apply_chat_template(x["prompt"], add_generation_prompt = True, tokenize = True)},
    batched = True,
).map(lambda x: {"length" : len(x["tokens"])}))["length"])

Map: 100%                               100/100 [00:00<00:00, 580.76 examples/s]
Map: 100%                               100/100 [00:00<00:00, 1105.59 examples/s]
976

max(test_data.map(
    lambda x: {"tokens" : tokenizer.apply_chat_template(x["prompt"], add_generation_prompt = True, tokenize = True)},
    batched = True,
).map(lambda x: {"length" : len(x["tokens"])}))["length"])

Map: 100%                               100/100 [00:00<00:00, 495.99 examples/s]
Map: 100%                               100/100 [00:00<00:00, 1119.31 examples/s]
976

max_prompt_length = 256

from trl import GRPOConfig, GRPOTrainer
training_args = GRPOConfig(
    learning_rate = 5e-6,

```

```

adam_beta1 = 0.9,
adam_beta2 = 0.99,
weight_decay = 0.1,
warmup_ratio = 0.1,
lr_scheduler_type = "cosine",
optim = "paged_adamw_8bit",
logging_steps = 1,
per_device_train_batch_size = 4,
gradient_accumulation_steps = 4, # Increase to 4 for smoother training
num_generations = 4, # Decrease if out of memory
max_prompt_length = max_prompt_length,
max_completion_length = max_seq_length - max_prompt_length,
# num_train_epochs = 1, # Set to 1 for a full training run
max_steps = 10,
save_steps = 5,
max_grad_norm = 0.1,
report_to = "none", # Can use Weights & Biases
output_dir = "outputs",
)

trainer = GRPOTrainer(
    model = model,
    processing_class = tokenizer,
    reward_funcs = [
        strict_format_reward_func,
        soft_format_reward_func,
        xmlcount_reward_func,
        correctness_reward_func,
    ],
    args = training_args,
    train_dataset = train_data,
    eval_dataset = eval_data,
)
trainer.train()

```

```

==((====))== Unsloth - 2x faster free finetuning | Num GPUs used = 1
\\  /| Num examples = 800 | Num Epochs = 1 | Total steps = 10
0^0/ \_/ \ Batch size per device = 4 | Gradient accumulation steps = 4
\  -___- / Data Parallel GPUs = 1 | Total batch size (4 x 4 x 1) = 16
"-_____" Trainable parameters = 32,768,000/4,000,000,000 (0.82% trained)
Unsloth: Will smartly offload gradients to save VRAM!
[10/10 1:57:04, Epoch 0/1]

```

Step	Training Loss	reward	reward_std	completion_length	kl	rewards / strict_format_reward_func	rewards / soft_format_reward_func	reward / xmlcount_reward_func
1	-0.000000	-0.124125	1.255097	363.500000	0.000000	0.000000	0.000000	0.000000
2	-0.000000	0.021812	2.304063	383.250000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.034750	0.817751	296.500000	0.000009	0.000000	0.000000	0.000000
4	0.000000	0.003062	1.870113	361.812500	0.000012	0.000000	0.000000	0.000000
5	0.000000	-0.211625	2.423703	372.375000	0.000013	0.000000	0.000000	0.000000
6	0.000000	-0.178750	1.447242	330.312500	0.000018	0.000000	0.000000	0.000000
7	0.000000	0.632375	0.728968	247.687500	0.000022	0.000000	0.000000	0.000000
8	0.000000	0.748938	1.517041	290.687500	0.000023	0.000000	0.000000	0.000000
9	0.000000	0.899750	1.238095	265.062500	0.000025	0.000000	0.000000	0.000000
10	0.000000	-0.084438	0.826160	320.375000	0.000018	0.000000	0.000000	0.000000

```

TrainOutput(global_step=10, training_loss=5.587790475658494e-07, metrics={'train_runtime': 8062.7507,
'train samples per second': 0.02, 'train steps per second': 0.001, 'total_flos': 0.0, 'train loss': 5.587790475658494e-

```

```
model.save_lora("/content/drive/MyDrive/grpo_saved_lora")
```

```

import re
import numpy as np
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from vllm import SamplingParams
sampling_params = SamplingParams(
    temperature=0.1,
    top_p=0.95,
    max_tokens=max_seq_length,
)

# Prediction function
def predict_answers(test_data, model, tokenizer, system_prompt, path=None):
    predictions = []

```



```

for i in range(min(50, len(test_data))): # Safe bound check
    try:
        text = tokenizer.apply_chat_template([
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": test_data[i]["question"]},
        ], tokenize=False, add_generation_prompt=False)

        output = model.fast_generate(
            text,
            sampling_params=sampling_params,
            lora_request=model.load_lora(path) if path else None,
        )[0].outputs[0].text

        extracted = extract_xml_answer(output)
        predicted_label = normalize_label(extracted if extracted else output)
        predictions.append(predicted_label)

    except Exception as e:
        print(f"[Error] Failed on index {i}: {e}")
        predictions.append("failed")

return predictions

# Evaluation function
def evaluate_predictions(y_true, y_pred):
    # Normalize labels to consistent lowercase format
    y_true = [normalize_label(y) for y in y_true]
    y_pred = [normalize_label(y) for y in y_pred]

    labels = ["Noncancer", "Cancer"]
    mapping = {label: idx for idx, label in enumerate(labels)}

    def map_func(x):
        return mapping.get(x, -1) # -1 will flag unknowns, ideally shouldn't occur

    y_true_mapped = np.vectorize(map_func)(y_true)
    y_pred_mapped = np.vectorize(map_func)(y_pred)

    # Calculate and print overall accuracy
    accuracy = accuracy_score(y_true=y_true_mapped, y_pred=y_pred_mapped)
    print(f'✅ Overall Accuracy: {accuracy:.3f}')

    # Per-class accuracy
    unique_labels = set(y_true_mapped)
    for label in unique_labels:
        label_indices = [i for i in range(len(y_true_mapped)) if y_true_mapped[i] == label]
        label_y_true = [y_true_mapped[i] for i in label_indices]
        label_y_pred = [y_pred_mapped[i] for i in label_indices]
        label_accuracy = accuracy_score(label_y_true, label_y_pred)
        print(f'🔵 Accuracy for label "{labels[label]}": {label_accuracy:.3f}')

    # Classification report
    print('\n📊 Classification Report:')
    print(classification_report(
        y_true=y_true_mapped,
        y_pred=y_pred_mapped,
        target_names=labels,
        labels=list(range(len(labels))),
        zero_division=0
    ))

    # Confusion matrix
    print('\n📊 Confusion Matrix:')
    print(confusion_matrix(
        y_true=y_true_mapped,
        y_pred=y_pred_mapped,
        labels=list(range(len(labels)))
    ))

#Before Pre-training
# 1. Generate predictions
predictions = predict_answers(test_data, model, tokenizer, system_prompt)
y_true = []
# 2. Extract true answers
for i in range(0,50):
    y_true.append(test_data[i]["answer"])

predictions, y_true = zip(*[
    (x, y) for x, y in zip(predictions, y_true) if x != "unknown" or x != "failed"
])

# Convert back to lists (zip returns tuples)

```




```
predictions = list(predictions)
y_true = list(y_true)


# 3. Evaluate
evaluate_predictions(y_true, predictions)
```

	Processed prompts: 100%	1/1 [00:24<00:00, 24.13s/it, est. speed input: 12.85 toks/s, output: 9.24 toks/s]
	Processed prompts: 100%	1/1 [00:15<00:00, 15.70s/it, est. speed input: 33.38 toks/s, output: 8.92 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.47s/it, est. speed input: 25.07 toks/s, output: 9.15 toks/s]
	Processed prompts: 100%	1/1 [00:21<00:00, 21.71s/it, est. speed input: 25.06 toks/s, output: 9.40 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.21s/it, est. speed input: 18.61 toks/s, output: 9.22 toks/s]
	Processed prompts: 100%	1/1 [00:27<00:00, 27.32s/it, est. speed input: 9.52 toks/s, output: 9.48 toks/s]
	Processed prompts: 100%	1/1 [00:26<00:00, 26.74s/it, est. speed input: 14.18 toks/s, output: 9.43 toks/s]
	Processed prompts: 100%	1/1 [01:21<00:00, 81.59s/it, est. speed input: 6.31 toks/s, output: 9.63 toks/s]
	Processed prompts: 100%	1/1 [01:32<00:00, 92.40s/it, est. speed input: 4.42 toks/s, output: 9.66 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.71s/it, est. speed input: 28.56 toks/s, output: 9.28 toks/s]
	Processed prompts: 100%	1/1 [00:17<00:00, 17.97s/it, est. speed input: 21.87 toks/s, output: 9.18 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.46s/it, est. speed input: 13.82 toks/s, output: 9.50 toks/s]
	Processed prompts: 100%	1/1 [00:16<00:00, 16.14s/it, est. speed input: 21.93 toks/s, output: 9.11 toks/s]
	Processed prompts: 100%	1/1 [00:20<00:00, 20.81s/it, est. speed input: 19.90 toks/s, output: 9.37 toks/s]
	Processed prompts: 100%	1/1 [00:23<00:00, 23.08s/it, est. speed input: 16.51 toks/s, output: 9.31 toks/s]
	Processed prompts: 100%	1/1 [00:57<00:00, 57.27s/it, est. speed input: 13.20 toks/s, output: 9.52 toks/s]
	Processed prompts: 100%	1/1 [00:17<00:00, 17.16s/it, est. speed input: 26.70 toks/s, output: 9.09 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.63s/it, est. speed input: 21.20 toks/s, output: 9.28 toks/s]
	Processed prompts: 100%	1/1 [00:17<00:00, 17.64s/it, est. speed input: 22.96 toks/s, output: 9.13 toks/s]
	Processed prompts: 100%	1/1 [00:20<00:00, 20.40s/it, est. speed input: 18.48 toks/s, output: 9.32 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.50s/it, est. speed input: 22.10 toks/s, output: 9.18 toks/s]
	Processed prompts: 100%	1/1 [01:08<00:00, 68.12s/it, est. speed input: 9.59 toks/s, output: 9.51 toks/s]
	Processed prompts: 100%	1/1 [00:14<00:00, 14.51s/it, est. speed input: 28.12 toks/s, output: 9.10 toks/s]
	Processed prompts: 100%	1/1 [00:17<00:00, 17.22s/it, est. speed input: 16.90 toks/s, output: 9.35 toks/s]
	Processed prompts: 100%	1/1 [00:31<00:00, 31.90s/it, est. speed input: 9.75 toks/s, output: 9.44 toks/s]
	Processed prompts: 100%	1/1 [00:23<00:00, 23.25s/it, est. speed input: 22.96 toks/s, output: 9.25 toks/s]
	Processed prompts: 100%	1/1 [01:27<00:00, 87.42s/it, est. speed input: 5.26 toks/s, output: 9.62 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.64s/it, est. speed input: 16.10 toks/s, output: 9.34 toks/s]
	Processed prompts: 100%	1/1 [00:17<00:00, 17.18s/it, est. speed input: 29.17 toks/s, output: 9.20 toks/s]
	Processed prompts: 100%	1/1 [01:20<00:00, 80.31s/it, est. speed input: 6.56 toks/s, output: 9.64 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.01s/it, est. speed input: 23.66 toks/s, output: 9.16 toks/s]
	Processed prompts: 100%	1/1 [00:15<00:00, 15.89s/it, est. speed input: 13.47 toks/s, output: 9.44 toks/s]
	Processed prompts: 100%	1/1 [01:13<00:00, 73.56s/it, est. speed input: 8.14 toks/s, output: 9.54 toks/s]
	Processed prompts: 100%	1/1 [01:30<00:00, 90.88s/it, est. speed input: 4.91 toks/s, output: 9.41 toks/s]
	Processed prompts: 100%	1/1 [00:20<00:00, 20.19s/it, est. speed input: 14.51 toks/s, output: 9.31 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.41s/it, est. speed input: 15.26 toks/s, output: 9.07 toks/s]
	Processed prompts: 100%	1/1 [01:22<00:00, 82.02s/it, est. speed input: 6.30 toks/s, output: 9.56 toks/s]
	Processed prompts: 100%	1/1 [00:14<00:00, 14.72s/it, est. speed input: 21.07 toks/s, output: 9.17 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.22s/it, est. speed input: 16.91 toks/s, output: 9.39 toks/s]
	Processed prompts: 100%	1/1 [00:17<00:00, 17.10s/it, est. speed input: 17.72 toks/s, output: 9.07 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.11s/it, est. speed input: 14.14 toks/s, output: 9.22 toks/s]
	Processed prompts: 100%	1/1 [00:21<00:00, 21.70s/it, est. speed input: 42.54 toks/s, output: 8.62 toks/s]
	Processed prompts: 100%	1/1 [00:16<00:00, 16.88s/it, est. speed input: 19.79 toks/s, output: 9.12 toks/s]
	Processed prompts: 100%	1/1 [01:07<00:00, 67.78s/it, est. speed input: 9.68 toks/s, output: 9.52 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.69s/it, est. speed input: 14.88 toks/s, output: 9.40 toks/s]
	Processed prompts: 100%	1/1 [01:28<00:00, 88.03s/it, est. speed input: 5.19 toks/s, output: 9.59 toks/s]
	Processed prompts: 100%	1/1 [01:18<00:00, 78.03s/it, est. speed input: 7.09 toks/s, output: 9.59 toks/s]
	Processed prompts: 100%	1/1 [00:20<00:00, 20.39s/it, est. speed input: 20.20 toks/s, output: 9.32 toks/s]
	Processed prompts: 100%	1/1 [01:26<00:00, 86.00s/it, est. speed input: 5.55 toks/s, output: 9.58 toks/s]


Processed prompts: 100%

1/1 [00:18<00:00, 18.64s/it, est. speed input: 20.92 toks/s, output: 9.23 toks/s]

 Overall Accuracy: 0.960
 Accuracy for label "Noncancer": 0.889
 Accuracy for label "Cancer": 1.000

 Classification Report:

	precision	recall	f1-score	support
Noncancer	1.00	0.89	0.94	18
Cancer	0.94	1.00	0.97	32
accuracy			0.96	50
macro avg	0.97	0.94	0.96	50
weighted avg	0.96	0.96	0.96	50

 Confusion Matrix:


```
[[16  2]
 [ 0 32]]
```

```
#After Pretraining
predictions = predict_answers(test_data, model, tokenizer, system_prompt, path="/content/drive/MyDrive/grpo_saved_lora")
y_true = []
# 2. Extract true answers
for i in range(0,50):
    y_true.append(test_data[i]["answer"])

predictions, y_true = zip(*[
    (x, y) for x, y in zip(predictions, y_true) if x != "unknown" or x != "failed"
])

# Convert back to lists (zip returns tuples)
predictions = list(predictions)
y_true = list(y_true)

# 3. Evaluate
evaluate_predictions(y_true, predictions)
```

	Processed prompts: 100%	1/1 [00:18<00:00, 18.37s/it, est. speed input: 16.87 toks/s, output: 7.95 toks/s]
	Processed prompts: 100%	1/1 [01:25<00:00, 85.32s/it, est. speed input: 6.14 toks/s, output: 9.11 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.25s/it, est. speed input: 24.05 toks/s, output: 8.57 toks/s]
	Processed prompts: 100%	1/1 [00:26<00:00, 26.08s/it, est. speed input: 20.86 toks/s, output: 8.70 toks/s]
	Processed prompts: 100%	1/1 [00:29<00:00, 29.05s/it, est. speed input: 11.67 toks/s, output: 8.95 toks/s]
	Processed prompts: 100%	1/1 [00:29<00:00, 29.70s/it, est. speed input: 8.75 toks/s, output: 8.92 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.55s/it, est. speed input: 20.43 toks/s, output: 8.73 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.31s/it, est. speed input: 26.66 toks/s, output: 8.54 toks/s]
	Processed prompts: 100%	1/1 [01:36<00:00, 96.95s/it, est. speed input: 4.21 toks/s, output: 9.21 toks/s]
	Processed prompts: 100%	1/1 [00:23<00:00, 23.86s/it, est. speed input: 23.60 toks/s, output: 8.38 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.74s/it, est. speed input: 19.91 toks/s, output: 8.71 toks/s]
	Processed prompts: 100%	1/1 [00:22<00:00, 22.92s/it, est. speed input: 11.74 toks/s, output: 8.90 toks/s]
	Processed prompts: 100%	1/1 [00:16<00:00, 16.93s/it, est. speed input: 20.91 toks/s, output: 8.62 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.84s/it, est. speed input: 20.86 toks/s, output: 8.77 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.52s/it, est. speed input: 19.51 toks/s, output: 8.66 toks/s]
	Processed prompts: 100%	1/1 [01:00<00:00, 60.29s/it, est. speed input: 12.54 toks/s, output: 9.04 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.41s/it, est. speed input: 24.88 toks/s, output: 8.58 toks/s]
	Processed prompts: 100%	1/1 [00:19<00:00, 19.23s/it, est. speed input: 20.54 toks/s, output: 8.79 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.27s/it, est. speed input: 22.17 toks/s, output: 8.59 toks/s]
	Processed prompts: 100%	1/1 [00:22<00:00, 22.36s/it, est. speed input: 16.86 toks/s, output: 8.85 toks/s]
	Processed prompts: 100%	1/1 [00:18<00:00, 18.38s/it, est. speed input: 23.45 toks/s, output: 8.11 toks/s]
	Processed prompts: 100%	1/1 [00:20<00:00, 20.67s/it, est. speed input: 31.60 toks/s, output: 8.52 toks/s]
	Processed prompts: 100%	1/1 [00:16<00:00, 16.61s/it, est. speed input: 24.56 toks/s, output: 8.31 toks/s]