Unsloth Fine-Tuning

Master Cheatsheet 2025

LoRA | QLoRA | PEFT | SFT | Complete Python Guide

■ What is Unsloth?

Unsloth = Ultra-fast LLM fine-tuning library. **2-5x faster** training with **80% less memory**. Free and open source!

Feature	Benefit
2-5x Faster Training	Custom CUDA kernels, optimized backprop
80% Less VRAM	Gradient checkpointing, 4-bit quantization
No Quality Loss	Same results as standard training
Free Tier	Works on Google Colab free GPUs
QLoRA Built-in	4-bit quantization + LoRA combined
Easy API	Simple, intuitive interface
Multiple Exports	GGUF, vLLM, HuggingFace, Ollama

■■ Installation

Environment	Command
Conda (Recommended)	conda create -n unsloth python=3.11 pytorch pytorch-cuda=12.1 -c pytorch -c nvidia
Pip (After Conda)	pip install unsloth
Colab/Kaggle	pip install unsloth
With All Extras	pip install "unsloth[all]"
Cutting Edge	pip install "unsloth @ git+https://github.com/unslothai/unsloth.git"

```
# Google Colab Installation
%%capture
!pip install unsloth
# Also install: pip install xformers trl peft accelerate bitsandbytes

# Required imports
from unsloth import FastLanguageModel
from unsloth import is_bfloat16_supported
from trl import SFTTrainer
from transformers import TrainingArguments
from datasets import load_dataset
import torch
```

■ Supported Models

Model Family	Supported Versions	Unsloth Model ID
Llama 3.3	70B	unsloth/Llama-3.3-70B-Instruct-bnb-4bit

Llama 3.2	1B, 3B	unsloth/Llama-3.2-3B-Instruct-bnb-4bit
Llama 3.1	8B, 70B, 405B	unsloth/Meta-Llama-3.1-8B-Instruct-bnb-4bit
Llama 3	8B, 70B	unsloth/llama-3-8b-Instruct-bnb-4bit
Mistral	7B v0.3, Nemo 12B	unsloth/mistral-7b-instruct-v0.3-bnb-4bit
Mistral Small	22B	unsloth/Mistral-Small-Instruct-2409-bnb-4bit
Qwen 2.5	0.5B to 72B	unsloth/Qwen2.5-7B-Instruct-bnb-4bit
Qwen 2.5 Coder	1.5B to 32B	unsloth/Qwen2.5-Coder-7B-Instruct-bnb-4bit
Gemma 2	2B, 9B, 27B	unsloth/gemma-2-9b-it-bnb-4bit
Phi-4	14B	unsloth/phi-4-bnb-4bit
Phi-3.5	3.8B	unsloth/Phi-3.5-mini-instruct-bnb-4bit
DeepSeek V3	685B MoE	unsloth/DeepSeek-V3-bnb-4bit
DeepSeek R1	1.5B to 70B	unsloth/DeepSeek-R1-Distill-Llama-8B-bnb-4bit

■ 4-bit models use 4x less VRAM. Add -bnb-4bit suffix for quantized versions!

■ LoRA & QLoRA Explained

Concept	Description
LoRA	Low-Rank Adaptation - trains small adapter matrices instead of full model
QLoRA	Quantized LoRA - 4-bit base model + LoRA adapters
Rank (r)	Size of adapter matrices. Higher = more capacity, more VRAM
Alpha	Scaling factor. Usually alpha = 2 * rank
Target Modules	Which layers to apply LoRA (q_proj, k_proj, v_proj, etc.)
Dropout	Regularization to prevent overfitting

Parameter	Recommended	Description
r (rank)	8, 16, 32, 64, 128	Adapter size. Start with 16 or 32
lora_alpha	16, 32, 64	Scaling. Usually 1-2x of rank
lora_dropout	0, 0.05, 0.1	0 for most cases
target_modules	See below	Layers to train
use_gradient_checkpointing	True	Saves VRAM
random_state	3407	For reproducibility

■ Target Modules by Model

Model	Target Modules
Llama/Mistral/Qwen	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj
Gemma	q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj

Phi	q_proj, k_proj, v_proj, dense, fc1, fc2
All (Recommended)	Use Unsloth auto-detection with get_peft_model()

■ Load Model with Unsloth

```
from unsloth import FastLanguageModel

# Configuration
max_seq_length = 2048 # Can go up to 8192 for longer context
dtype = None # Auto-detect (Float16 for Tesla T4, BFloat16 for Ampere+)
load_in_4bit = True # Use 4-bit quantization (QLoRA)

# Load model and tokenizer
model, tokenizer = FastLanguageModel.from_pretrained(
model_name='unsloth/Llama-3.2-3B-Instruct-bnb-4bit',
max_seq_length=max_seq_length,
dtype=dtype,
load_in_4bit=load_in_4bit,
# token='hf_...', # Optional: for gated models
)

print(f'Model loaded! VRAM: {torch.cuda.memory_allocated()/1e9:.2f} GB')
```

■ PEFT / LoRA Configuration

```
# Add LoRA adapters to model
model = FastLanguageModel.get_peft_model(
model,
r=16, # LoRA rank (8, 16, 32, 64, 128)
target_modules=[
'q_proj', 'k_proj', 'v_proj', 'o_proj',
'gate_proj', 'up_proj', 'down_proj',
],
lora_alpha=16, # Scaling factor
lora_dropout=0, # 0 = optimized, no dropout
bias='none', # 'none' for speed
use_gradient_checkpointing='unsloth', # 30% less VRAM
random_state=3407,
use_rslora=False, # Rank-stabilized LoRA
loftq_config=None, # LoftQ quantization
)

# Check trainable parameters
model.print_trainable_parameters()
# Output: trainable params: 41,943,040 || all params: 3,253,145,600 || 1.29%
```

■ Dataset Preparation

Format	Description	Use Case
Alpaca	instruction, input, output	Instruction following
ShareGPT	conversations: [{from, value}]	Multi-turn chat
ChatML	messages: [{role, content}]	OpenAI format
Completion	text only	Text completion
Custom	Any format with formatting_func	Custom templates

■ Alpaca Format (Most Common)

```
# Alpaca prompt template
alpaca_prompt = '''Below is an instruction that describes a task, paired with an input.
Write a response that appropriately completes the request.

### Instruction:
{instruction}

### Input:
{input}

### Response:
{output}'''

# Format dataset
def formatting_prompts_func(examples):
instructions = examples['instruction']
inputs = examples['input']
outputs = examples['output']
texts = []
for inst, inp, out in zip(instructions, inputs, outputs):
text = alpaca_prompt.format(instruction=inst, input=inp, output=out)
texts.append(text + tokenizer.eos_token)
return {'text': texts}

# Load and format dataset
from datasets import load_dataset
dataset = load_dataset('yahma/alpaca-cleaned', split='train')
dataset = dataset.map(formatting_prompts_func, batched=True)
```

■ ChatML / Conversation Format

```
# For chat/instruct models (Llama 3, Qwen, etc.)
from unsloth.chat_templates import get_chat_template

tokenizer = get_chat_template(
tokenizer,
chat_template='llama-3.1', # or 'chatml', 'mistral', 'gemma', 'qwen-2.5'
)

def formatting_func(examples):
convos = examples['conversations']
texts = []
for convo in convos:
text = tokenizer.apply_chat_template(
convo, tokenize=False, add_generation_prompt=False
)
texts.append(text)
return {'text': texts}

dataset = load_dataset('philschmid/guanaco-sharegpt-style', split='train')
dataset = dataset.map(formatting_func, batched=True)
```

■ SFT Training (Supervised Fine-Tuning)

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
model=model,
tokenizer=tokenizer,
train_dataset=dataset,
dataset_text_field='text',
max_seq_length=max_seq_length,
dataset_num_proc=2,
packing=False, # True for short sequences
args=TrainingArguments(
per_device_train_batch_size=2,
gradient_accumulation_steps=4, # Effective batch = 2*4 = 8
warmup_steps=5,
num_train_epochs=1, # Or use max_steps
# max_steps=60,
learning_rate=2e-4,
fp16=not is_bfloat16_supported(),
bf16=is_bfloat16_supported(),
logging_steps=1,
optim='adamw_8bit',
weight_decay=0.01,
lr_scheduler_type='linear',
seed=3407,
output_dir='outputs',
report_to='none', # or 'wandb'
),
)
```

Parameter	Recommended	Description
per_device_train_batch_size	2	Batch size per GPU
gradient_accumulation_steps	4	Accumulate before update
learning_rate	2e-4 to 5e-5	Start with 2e-4
num_train_epochs	1-3	Full passes over data
max_steps	-1 or N	-1 for epochs, N for steps
warmup_steps	5-100	LR warmup
optim	adamw_8bit	8-bit Adam saves VRAM
weight_decay	0.01	Regularization
lr_scheduler_type	linear, cosine	LR decay schedule
packing	True/False	Pack short sequences together

```
# Show GPU stats before training
gpu_stats = torch.cuda.get_device_properties(0)
print(f'GPU: {gpu_stats.name}, VRAM: {gpu_stats.total_memory/1e9:.2f} GB')

# Train!
trainer_stats = trainer.train()

# Show training stats
print(f'Training time: {trainer_stats.metrics["train_runtime"]/60:.2f} min')
print(f'Peak VRAM: {torch.cuda.max_memory_allocated()/1e9:.2f} GB')
```

■ Inference (Using the Model)

```
# Enable fast inference mode
FastLanguageModel.for_inference(model)

# Method 1: Direct generation
inputs = tokenizer(
[alpaca_prompt.format(
instruction='Write a poem about AI',
input='',
output='' # Leave empty for generation
)],
return_tensors='pt'
).to('cuda')

outputs = model.generate(
**inputs,
max_new_tokens=256,
use_cache=True,
temperature=0.7,
top_p=0.9,
)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))

# Method 2: Streaming
from transformers import TextStreamer
streamer = TextStreamer(tokenizer)
outputs = model.generate(**inputs, streamer=streamer, max_new_tokens=256)
```

■ Save & Export Models

Format	Use Case	Method
LoRA Adapters	Small file, needs base model	save_pretrained_gguf
Merged 16-bit	Full model, HuggingFace compatible	save_pretrained_merged
Merged 4-bit	Quantized full model	save_pretrained_merged (4bit)
GGUF (Q4_K_M)	Ollama, llama.cpp	save_pretrained_gguf
GGUF (Q8_0)	Higher quality GGUF	save_pretrained_gguf
GGUF (f16)	Full precision GGUF	save_pretrained_gguf
vLLM	Fast inference server	Merged model
Push to Hub	Share on HuggingFace	push_to_hub()

```
# Save LoRA adapters only (small file)
model.save_pretrained('lora_model')
tokenizer.save_pretrained('lora_model')

# Save merged model (16-bit)
model.save_pretrained_merged('merged_model', tokenizer, save_method='merged_16bit')

# Save merged model (4-bit quantized)
model.save_pretrained_merged('merged_4bit', tokenizer, save_method='merged_4bit_forced')

# Save as GGUF for Ollama/llama.cpp
model.save_pretrained_gguf('gguf_model', tokenizer, quantization_method='q4_k_m')
# Options: q4_k_m, q5_k_m, q8_0, f16

# Push to HuggingFace Hub
model.push_to_hub('your-username/model-name', token='hf_...')
tokenizer.push_to_hub('your-username/model-name', token='hf_...')

# Push GGUF to Hub
model.push_to_hub_gguf('your-username/model-gguf', tokenizer, quantization_method='q4_k_m')
```

■ Ollama Integration

```
# After saving as GGUF, create Modelfile
# Modelfile contents:
FROM ./gguf_model/unsloth.Q4_K_M.gguf

TEMPLATE '''Below is an instruction. Write a response.
### Instruction:
{{ .Prompt }}
### Response:
'''

PARAMETER stop "### Instruction:"
PARAMETER temperature 0.7

# Create and run with Ollama
# ollama create mymodel -f Modelfile
# ollama run mymodel "Your prompt here"
```

■ Complete Training Script

```
# === COMPLETE UNSLOTH FINE-TUNING SCRIPT ===
from unsloth import FastLanguageModel, is_bfloat16_supported
from trl import SFTTrainer
from transformers import TrainingArguments
from datasets import load_dataset
import torch

# 1. Configuration
max_seq_length = 2048
model_name = 'unsloth/Llama-3.2-3B-Instruct-bnb-4bit'

# 2. Load Model
model, tokenizer = FastLanguageModel.from_pretrained(
model_name=model_name, max_seq_length=max_seq_length,
dtype=None, load_in_4bit=True
)

# 3. Add LoRA
model = FastLanguageModel.get_peft_model(
model, r=16, lora_alpha=16, lora_dropout=0,
target_modules=['q_proj','k_proj','v_proj','o_proj',
'gate_proj','up_proj','down_proj'],
use_gradient_checkpointing='unsloth'
)

# 4. Prepare Dataset (see dataset section)
# 5. Train with SFTTrainer (see training section)
# 6. Save model (see export section)
```

■ Best Practices

Category	Best Practice
VRAM	Start with 4-bit models, increase batch size if VRAM allows
Rank	Start with r=16, increase to 32/64 if underfitting
Learning Rate	Start with 2e-4, reduce if loss spikes
Epochs	Start with 1-3 epochs, watch for overfitting
Dataset	Quality over quantity. 1000 good examples beat 100k bad ones

Validation	Always keep a validation set to monitor overfitting
Gradient Checkpointing	Always use use_gradient_checkpointing="unsloth"
Packing	Enable packing=True for short sequences to speed up training
EOS Token	Always append tokenizer.eos_token to training examples
Chat Template	Use correct chat template for instruct models
Export	Save both LoRA adapters AND merged model for flexibility
Testing	Test inference before and after training to verify improvement

■ VRAM Requirements

Model Size	4-bit (QLoRA)	16-bit (LoRA)	GPU Examples
1B-3B	4-6 GB	8-12 GB	T4, RTX 3060
7B-8B	6-10 GB	16-24 GB	RTX 3080, A10
13B	10-14 GB	28-32 GB	RTX 4090, A10
34B-40B	20-24 GB	48-64 GB	A100 40GB
70B	36-48 GB	140+ GB	A100 80GB, H100

■ Free Colab (T4 16GB) can fine-tune 3B-8B models with 4-bit!

Created by Manoj | The AI Dude Tamil ■

YouTube: The AI Dude Tamil | Master AI, Automation & Prompt Engineering

■ Fine-tune LLMs 2-5x faster with Unsloth!