Installation and Setup

Onboarding Guide to Serverless GPU CLI: SGCLI

Ben Hansen
Last updated: Jan 23, 2026

Google Drive folder with wheel

Overview

The Serverless GPU CLI (`sgcli`) is a command-line tool for submitting and managing training jobs on Databricks Serverless GPU compute directly from your terminal, without using notebooks.

Databricks serverless GPU compute provides on-demand A10 and H100 accelerators for deep learning training and fine-tuning workloads, integrated with notebooks, Unity Catalog, and MLflow.

Please sign up for the **Serverless GPU Compute API Remote H100s Private Preview** to use this feature.

Installation and setup

Setup your databricks credentials

See https://docs.databricks.com/aws/en/dev-tools/cli/authentication for general Databricks authentication details.

Step1. Install databricks CLI if you haven't already done so (On MacOs): brew install databricks.
Step2. Authenticate to your workspace

```
Shell
databricks auth login --host https://mycompany.cloud.databricks.com
```

This should take you through the Oauth flow and create a `databrickscfg` file if it doesn't already exist:
Databricks configuration profiles

The CLI tool will authenticate to Databricks using your credentials in the file. The file's default location is ~/.databrickscfg as documented in the link above.

```
None
[DEFAULT]
host      = https://my-workspace.databricks.com
auth_type = databricks-cli

[dev]
host      = https://dev-workspace.com/
auth_type = databricks-cli
```

Step3. Set the environment variable `DATABRICKS_CONFIG_PROFILE` to the desired profile

```
Shell
export DATABRICKS_CONFIG_PROFILE=dev
```

Which would change your profile to dev instead of DEFAULT in the current terminal session.

Install the wheel

You can find a stable wheel here in gdrive.

1. Create or use an existing python venv and install the wheel downloaded from this Google Drive.

```
Shell
python3.12 -m venv try-sgcli
source try-sgcli/bin/activate

uv pip install
/path/to/wheel/databricks_serverless_gpu_cli-0.0.1-py3-none-any.whl
--force-reinstall
```

Execute a workload

You will need to do a few more things to run a workload.
The main pieces are:

- workload.yaml - This is how to describe your training workload to Databricks. Go to Workload YAML Reference section for more information
- requirements.yaml - This describes your python dependencies and base environment
- Command string - This is your launch command(s)
- train.py - This is your actual training code. It could be a self contained script or part of a larger training library. Generally you reference it in the command string, i.e. `command: torchrun <torch-run-args> train.py`

The example below illustrates a simple example using torchrun to launch a python script.
Got to the Training Examples section for more complex examples using sgcli.

1. Create workload.yaml

Create a yaml file that specifies the workload. This is the primary surface in which you describe your workload to Databricks Serverless GPU.

```
None
experiment_name: helloworld-torchrun
environment:
  env_variables:
    NCCL_DEBUG: "INFO"
  dependencies: requirements.yaml
compute:
  gpus: 2
  gpu_type: a10
max_retries: 0
workspace:
  host: https://my-workspace.databricks.com # change to your workspace
command: |-
  torchrun \
    --nnodes=2 \
    --nproc_per_node=1 \
    --node_rank=$NODE_RANK \
    --master_addr=$MASTER_ADDR \
    --master_port=$MASTER_PORT \
    /Workspace/path/to/train.py  # CHANGE THIS TO YOUR OWN PATH
```

You can use this simple train.py for example

See YAML Reference in the user manual for details.

2. Define requirements.yaml

This file defines that the training job will be running version 4 of the base environment and installs the latest version of the mlflow package.

```
None
version: "4"
dependencies:
  - mlflow
```

Optionally, if you already have a requirements.txt you can also just reference it from your Workspace. For details, refer here.

```
None
version: "4"
dependencies:
  - -r "/Workspace/path/to/requirements.txt" # CHANGE THIS TO YOUR OWN PATH
```

3. Submit the workload

Activate your venv (if not already active) and submit your workload with sgcli

```
None
source try-sgcli/bin/activate

sgcli run -f workload.yaml
```

See SGCLI Commands sectionl for details on commands and flags.

SGCLI Commands

Command Reference

Brief reference for Serverless GPU CLI (`sgcli`) commands.

For detailed help on any command, use:

```
sgcli -h                    # Show all commands
sgcli <command> -h          # Show help for specific command
sgcli get <resource> -h     # Show help for get subcommands
```

Quick Reference

Submit a Training Job

```
sgcli run --file train.yaml
```

Common options:

- `--watch` - Monitor job and stream logs until completion
- `--override KEY=VALUE` - Override YAML parameters (e.g., `compute.gpus=32`)
- `-p PROFILE` - Use specific Databricks profile from `~/.databrickscfg`

Examples:

```
# Submit and wait for completion
sgcli run --file train.yaml --watch

# Override parameters
sgcli run --file train.yaml --override compute.gpus=32 timeout_minutes=120
```

Get Job Status

```
sgcli get status <run-id>
```

View Logs

```
sgcli get logs <run-id>
```

Options:
- `--rank N` - View logs from specific GPU rank (default: 0)
- `--lines N` - Show only last N lines
- `--debug` - Download and analyze logs from all nodes, filtering for errors

Examples:

```
# View logs from rank 0
sgcli get logs 388840544681214

# View logs from rank 2
sgcli get logs 388840544681214 --rank 2

# Debug mode (all nodes, errors only)
sgcli get logs 388840544681214 --debug
```

List Your Jobs

`sgcli get runs`

Options:

- `--limit N` - Maximum number of runs to show (default: 20)
- `--all` - Show all runs including completed (default: active only)
- `--experiment NAME` - Filter by experiment name (supports regex)
- `-p PROFILE` - Use specific Databricks profile
- `--host URL` - Specify workspace URL

Examples:

```
# Show last 10 active runs
sgcli get runs --limit 10

# Show all runs including completed
sgcli get runs --all

# Filter by experiment name
sgcli get runs --experiment "qwen.*"
```

Cancel a Job

`sgcli cancel <run-id>`

Example:

```
sgcli cancel 388840544681214
```

Debugging Tool Errors

Use verbose logging to help diagnose errors.

- `-v, --verbose` - Increase logging verbosity

Example:

```
sgcli -v run -f train.yaml
```

Workload YAML Reference

YAML Configuration Reference

Reference for Serverless GPU CLI (`sgcli`) training job configurations.

Quick Start

Minimal Configuration

```yaml
experiment_name: my-training
environment:
  dependencies: requirements.yaml
compute:
  gpus: 1
  gpu_type: a10
workspace:
  host: https://my-workspace.databricks.com
command: echo “Hello World”
```

Run Command

```
sgcli run --file train.yaml
```

Core Concepts

Required Fields

Every training configuration needs four essential components:

1. **experiment_name**: A name for your MLflow experiment
2. **environment**: Environment configuration with dependencies
3. **compute**: GPU resources (type and count)
4. **command**: Your bash launch commands

Your First Training Job

```yaml
experiment_name: simple-training
environment:
  dependencies: requirements.yaml
compute:
  gpus: 8
  gpu_type: h100
workspace:
  host: https://my-workspace.databricks.com
command: torchrun --nproc_per_node=8 train.py
```

This configuration:
- Installs dependencies from `requirements.yaml`
- Logs results to MLflow experiment “simple-training”
- Runs `train.py` via torchrun with 8 H100 GPUs.

Common Use Cases

Adding Environment Variables

```yaml
experiment_name: training-with-env
environment:
  dependencies: requirements.yaml
  env_variables:
    BATCH_SIZE: "32"
    LEARNING_RATE: "0.001"
compute:
  gpus: 8
  gpu_type: h100
workspace:
  host: https://my-workspace.databricks.com
code_source:
  type: snapshot
  snapshot:
    git_branch: main
    repo_path: /home/username/repo
command: torchrun --nproc_per_node=8 train.py
```

Using Secrets (API Keys, Tokens)

```yaml
experiment_name: training-with-secrets
environment:
  dependencies: requirements.yaml
  env_variables_secrets:
    HF_TOKEN: "secrets/hf_token"
    WANDB_API_KEY: "secrets/wandb"
compute:
  gpus: 8
  gpu_type: h100
workspace:
  host: https://my-workspace.databricks.com
code_source:
  type: snapshot
  snapshot:
    git_branch: main
    repo_path: /home/username/repo
command: torchrun --nproc_per_node=8 train.py
```

Note: Secrets use the format `"scope/key"` and must be configured in Databricks Secrets. See https://docs.databricks.com/aws/en/security/secrets/ for details.

Access Note: If sharing yaml templates, other users must create their own secrets or have access to the referenced secret. This is also described in the link above.

Working with Git Repositories

Package your local git repository for training:

```yaml
experiment_name: github-training
environment:
  dependencies: requirements.yaml
compute:
  gpus: 8
  gpu_type: h100
workspace:
  host: https://my-workspace.databricks.com
code_source:
  type: snapshot
  snapshot:
    git_branch: main
    git_commit: abc1234567
    repo_path: /home/username/repo
command: torchrun --nproc_per_node=8 train.py
```

Key fields:
- `repo_path`: (Required) Local path to your git repository
- `git_branch`: (Optional) Branch name (currently checkout out commit will be used if not provided)
- `git_commit`: (Optional) Specific commit SHA

The repository will be placed at $HOME

```
# Navigate to repository directory and execute training script

cd $HOME/my_repository
./start_training.sh
```

Advanced Features

Custom Hyperparameters

Pass structured configuration to your training script:

```yaml
experiment_name: parameterized-training
environment:
  dependencies: requirements.yaml
compute:
  gpus: 8
  gpu_type: h100
workspace:
```

```yaml
  host: https://my-workspace.databricks.com
code_source:
  type: snapshot
  snapshot:
    git_branch: main
    repo_path: /home/username/repo
command: torchrun --nproc_per_node=8 train.py
parameters:
  model:
    name: "gpt2"
    hidden_size: 768
  training:
    batch_size: 32
    learning_rate: 0.0001
```

Reading parameters in your script:

```python
import os
import yaml

with open(os.environ['HYPERPARAMETERS_PATH']) as f:
    params = yaml.safe_load(f)

learning_rate = params['training']['learning_rate']
model_name = params['model']['name']
```

Job Reliability

```yaml
experiment_name: reliable-training
environment:
  dependencies: requirements.yaml
compute:
  gpus: 8
  gpu_type: h100
workspace:
  host: https://my-workspace.databricks.com
code_source:
  type: snapshot
  snapshot:
    git_branch: main
    repo_path: /home/username/repo
command: torchrun --nproc_per_node=8 train.py
max_retries: 2
timeout_minutes: 90
```

If the workload fails for any reason it will be retried twice. The Job will timeout and stop after 90 minutes.

Complex Research Workflow Example

Large-scale training with all advanced features:

```yaml
experiment_name: qwen3vl-finetuning
workspace:
  host: https://adb-4599328495546933.13.azuredatabricks.net/
environment:
  dependencies: requirements.yaml
  env_variables:
    NCCL_DEBUG: "INFO"
compute:
  gpus: 64
  gpu_type: h100
workspace:
  host: https://my-workspace.databricks.com
code_source:
  type: snapshot
  snapshot:
    git_branch: qwen_sft_test
    git_commit: abc1234567
    repo_path: /home/username/company_repo
max_retries: 2
parameters:
  seed: 17
  variables:
    model_name: "Qwen/Qwen2-VL-7B-Instruct"
    context_len: 2048
  model:
    name: hf_multimodal_lm
    pretrained: true
    init_device: mixed
    use_auth_token: true
    attn_implementation: sdpa
    pretrained_model_name_or_path: ${paramters.variables.model_name} #
minimizes value duplication in yaml
  training:
    batch_size: 32
    learning_rate: 0.0001

command: |-

      torchrun \
      --nproc_per_node=8 \
      --nnodes=$WORLD_SIZE \
      --node_rank=$NODE_RANK \
      --master_addr=$MASTER_ADDR \
      --master_port=$MASTER_PORT \
      train.py
```

Reference

Complete Field Reference

Required Fields

Field	Type	Description	Example
`experiment_name`	string	Experiment name for MLflow	`"my-training-job"`
`environment.dependencies`	string	Path to requirements.yaml	`"requirements.yaml"`
`compute.gpus`	int	Number of GPUs	`1`, `4`, `8`
`compute.gpu_type`	string	GPU type	`"h100"`, `"a10"`
`code_source`	dict	Code source configuration	See below
`command`	string	Bash commands to launch training	`torchrun --nproc_per_node=8 train.py`

Code Source (Required):

```yaml
code_source:
  type: snapshot
  snapshot:
    git_branch: main
    git_commit: abc1234567  # Optional but recommended
    repo_path: /home/username/repo
```

Supported GPU Types:
- `"h100"` or `"H100"` (8 GPUs/node)
- `"a10"`, `"A10"`, or `"a10g"` (1 GPU/node)

Optional Fields

Workspace Configuration

Target workspace to use. You must have access to the workspace and have authenticated to it via

```
None
databricks auth login --host https://my-workspace.databricks.com
```

If not specified, the default workspace in your databrickscfg file will be used.

```yaml
workspace:
  host: https://my-workspace.databricks.com
```

Environment Configuration

Configure your Python environment and variables:

```yaml
environment:
  dependencies: requirements.yaml
  env_variables:
    BATCH_SIZE: "32"
  env_variables_secrets:
    HF_TOKEN: "my_scope/hf_token"
```

Requirements file format:

```yaml
version: "4"  # Optional: Runtime version "3" or "4" (default: "4")
dependencies:
  - torch==2.1.0
  - transformers==4.35.0
```

The `version` field is optional and defaults to `"4"` if not specified.

Compute Configuration

```yaml
compute:
  gpus: 8
  gpu_type: h100
```

Code Source Configuration

The `code_source` field specifies where your training code comes from using a discriminated union:

```yaml
code_source:
  type: snapshot
  snapshot:
    git_branch: main                          # Optional (will use whatever branch is currently checked out if not provided)
    git_commit: abc1234567                    # Optional (will use HEAD of specified branch if not provided)
    repo_path: /home/username/repo            # REQUIRED: Local path to git repo
```

Fields:
- `type`: Must be `"snapshot"`
- `snapshot.git_branch`: Branch name
- `snapshot.git_commit`: Specific commit SHA (optional but recommended for

reproducibility)
- `snapshot.repo_path`: Local path to your git repository

Note: The snapshot creates a package from your local git repository. Changes must be committed.

Custom Parameters

```yaml
parameters:                 # Passed via HYPERPARAMETERS_PATH env var
  model:
    name: "gpt2"
    hidden_size: 768
  training:
    batch_size: 32
```

MLflow Run Name

```yaml
run_name: "experiment-001-baseline"  # Custom display name
```

Path Resolution

All paths in the workload YAML are relative to the workload YAML unless they are absolute paths.

Example:

Folder structure:

```
/home/username/my-project/
├── train.yaml
├── requirements.yaml
└── scripts/
    └── train.py
```

YAML configuration:

```yaml
experiment_name: my-training
environment:
  dependencies: requirements.yaml  # Relative to train.yaml
compute:
  gpus: 8
  gpu_type: h100
code_source:
  type: snapshot
  snapshot:
    git_branch: main
    repo_path: /home/username/my_project # Absolute path
# Repo is placed at $HOME so train.py is referenced with the repo
command: torchrun --nproc_per_node=8 $HOME/my-project/scripts/train.py
```

Training Examples

Training Examples

Axolotl Llama 70B

The following example training Llama 70B using axolotl on 16 H100 GPU's.

Workload Yaml

```
None
experiment_name: axolotl-llama-70b

environment:
  dependencies: requirements.yaml
  env_variables_secrets:
    HF_TOKEN: "hf/hugging-face-read" # change this to your own hugging face
token
compute:
  gpus: 16
  gpu_type: h100

max_retries: 0
workspace:
  host: https://my-workspace.databricks.com # change to your workspace

command: |-

  # path to the accelerate config
  ACCEL_CONFIG="/Workspace/path/to/accelerate_config.yaml"

  # path to the axolotl config
  TRAIN_CONFIG="/Workspace/path/to/axolotl-example-fft-70b-liger-fsdp.yaml"

  accelerate launch \
    --config_file "$ACCEL_CONFIG" \
    --machine_rank=$NODE_RANK \
    --num_machines 2 \
    --num_processes $WORLD_SIZE \
    --main_process_ip $MASTER_ADDR \
    --main_process_port $MASTER_PORT \
    -m axolotl.cli.train "$TRAIN_CONFIG"
```

requirements.yaml

```
None
version: "4"
dependencies:
  - packaging
  - setuptools
  - wheel
  - ninja
  - mlflow>=3.6

  # Axolotl with flash-attn and deepspeed (no build isolation)
  - --no-build-isolation
  - axolotl[flash-attn,deepspeed]==0.12.2

  # Flash attention wheel (no deps - specific CUDA/torch version)
  - --no-deps
  -
https://github.com/Dao-AILab/flash-attention/releases/download/v2.7.4.post1/fla
sh_attn-2.7.4.post1+cu12torch2.6cxx11abiFALSE-cp312-cp312-linux_x86_64.whl
```

Training configs

Place these files in your Databricks Workspace and update the command in the Workload yaml to reference their paths.

accelerate_config.yaml

```
None
compute_environment: LOCAL_MACHINE
debug: false
distributed_type: FSDP
downcast_bf16: 'no'
machine_rank: 0 # Set to 0 for the main machine, increment by one for other
machines
main_process_ip: 10.0.0.4 # Set to main machine's IP
main_process_port: 5000
main_training_function: main
mixed_precision: bf16
num_machines: 2 # Change to the number of machines
```

```
num_processes: 8 # That's the total number of GPUs, (for example: if you have 2
machines with 4 GPU, put 8)
rdzv_backend: static
same_network: true
tpu_env: []
tpu_use_cluster: false
tpu_use_sudo: false
use_cpu: false
```

axolotl-example-fft-70b-liger-fsdp.yaml
NOTE: Some values like UC Volume paths need to be changed.

```
None
# Axolotl Configuration: Llama-3.3-70B FFT with SDPA
# Optimized based on Qwen32B benchmarks (RECOMMENDED)
#
# Key features:
# - Full Fine-Tuning (FFT) - 100% of 70.6B parameters
# - SDPA (PyTorch native, 7.2% faster than Flash Attention 2)
# - Liger kernel optimizations (RoPE, RMSNorm, GLU, CrossEntropy)
# - FSDP with CPU offloading
# - Optimized dataloader (16 workers, 256 prefetch)
# - Production dataset: FineTome-100k

base_model: meta-llama/Llama-3.3-70B-Instruct
model_type: llama

# ==========================================
# LIGER KERNEL OPTIMIZATIONS
# ==========================================

plugins:
  - axolotl.integrations.liger.LigerPlugin

liger_rope: true
liger_rms_norm: true
liger_glu_activation: true
liger_fused_linear_cross_entropy: true

# ==========================================
# DATASET CONFIGURATION
```

```yaml
# ==========================================

chat_template: llama3

datasets:
  - path: mlabonne/FineTome-100k-dedup
    type: chat_template
    split: train[:20%]  # 20,000 examples for testing
    field_messages: conversations
    message_property_mappings:
      role: from
      content: value

dataset_prepared_path: last_run_prepared
val_set_size: 0.02  # 2% validation

# ==========================================
# OUTPUT CONFIGURATION
# ==========================================

output_dir: /Volumes/path/to/output # CHANGEME

# ==========================================
# TRAINING HYPERPARAMETERS
# ==========================================

# Sequence length (OOM at 2048 was caused by missing activation offloading)
# With activation_offloading enabled, can use full 2048 sequence length
# Activations are the largest memory consumer (~60+ GB per GPU at 2048 seq)
sequence_len: 2048
sample_packing: true
pad_to_sequence_len: true
eval_sample_packing: false          # avoids the error you hit with tiny eval
splits

# ==========================================
# ACTIVATION OFFLOADING (CRITICAL FOR 70B)
# ==========================================
# This was the missing piece that caused OOM!
# Offloads activation tensors to CPU during backward pass
# Required for 70B models at sequence length > 1024
activation_offloading: true

# Batch size and gradient accumulation
```

```yaml
# Optimized for 32 H100 GPUs: 1 × 8 × 32 = 256 effective batch
micro_batch_size: 1
gradient_accumulation_steps: 32

# Training duration
num_epochs: 1

# Optimizer and learning rate
optimizer: adamw_torch_fused
lr_scheduler: cosine
learning_rate: 2e-5
warmup_ratio: 0.1

# Mixed precision
bf16: true
tf32: false

# ============================================
# SDPA ATTENTION CONFIGURATION
# ============================================

# Use PyTorch native SDPA (Scaled Dot-Product Attention)
# This is 7.2% faster than Flash Attention 2 with FSDP + CPU offload on H100
# Based on Qwen32B benchmarks: SDPA 34.23s/step vs FA2 36.87s/step

flash_attention: false
attn_implementation: sdpa
sdpa_attention: true

# Note: SDPA is PyTorch native and optimized for:
# - FSDP + CPU offloading workflows
# - H100 Tensor Cores
# - Sequence lengths < 4K
# For sequences > 8K, consider Flash Attention 2 instead

# ============================================
# MEMORY OPTIMIZATIONS
# ============================================

gradient_checkpointing: true
gradient_checkpointing_kwargs:
  use_reentrant: false

# ============================================
```

```
# FSDP CONFIGURATION
# ==========================================

# FSDP strategy (full shard + auto wrap + CPU offload)
fsdp_version: 2

fsdp:
  - full_shard
  - auto_wrap

fsdp_config:
  offload_params: true                       # was fsdp_offload_params
  cpu_ram_efficient_loading: true           # was
fsdp_cpu_ram_efficient_loading
  auto_wrap_policy: TRANSFORMER_BASED_WRAP  # was fsdp_auto_wrap_policy
  transformer_layer_cls_to_wrap: LlamaDecoderLayer
  state_dict_type: FULL_STATE_DICT          # same meaning
  reshard_after_forward: true               # replaces
fsdp_sharding_strategy=FULL_SHARD
  activation_checkpointing: true            # was fsdp_activation_checkpointing
  sync_module_states: true
  limit_all_gathers: true
  backward_prefetch: BACKWARD_PRE
  forward_prefetch: true

# ==========================================
# DATALOADER OPTIMIZATION
# ==========================================
# KEY LEARNINGS from Qwen32B:
# - Dataloader optimization = 2× speedup (most critical!)
# - 16 workers, 256 prefetch, 192 dataset processes

dataloader_num_workers: 16         # Increased from 4 to 16
dataloader_prefetch_factor: 256    # NEW: Aggressive prefetching
dataloader_pin_memory: false       # False with CPU offload
dataset_processes: 192             # NEW: Parallel dataset processing

# ==========================================
# LOGGING AND CHECKPOINTING
# ==========================================

logging_steps: 10
evals_per_epoch: 2
saves_per_epoch: 1
```

```
save_strategy: "epoch"

# ============================================
# SPECIAL TOKENS
# ============================================

special_tokens:
  pad_token: <|finetune_right_pad_id|>
  eos_token: <|eot_id|>

# ============================================
# WANDB (Optional)
# ============================================
wandb_mode: disabled
wandb_project:
wandb_entity:
wandb_watch:
wandb_name:
wandb_log_model:

# ============================================
# MLflow (Recommended)
# ============================================
use_mlflow: true
mlflow_tracking_uri: databricks        # or your Databricks / GenAI tracking
URI
mlflow_run_name: fft-70b-liger-fsdp-resume
hf_mlflow_log_artifacts: false # Disable this due to long upload time

# ============================================
# ADDITIONAL SETTINGS
# ============================================

# Memory management
max_memory_mb: 75000  # 75GB per GPU (for H100 80GB)
pad_to_sequence_len: false

# Weight decay
weight_decay: 0.0

# Resume from checkpoint (optional)
resume_from_checkpoint: /Volumes/path/to/checkpoint # CHANGEME
```

Requirements Yaml Reference

Requirements Yaml Reference

The requirements yaml, i.e. the field reference in the workload dependencies defines the python libraries that will be installed for your workload.

```
None
environment:
  dependencies: requirements.yaml
```

The yaml is based on the officiant Base Environment Specification here: https://docs.databricks.com/aws/en/admin/workspace-settings/base-environment#example-environment-specification

Example

In the following example:

1. the Base Environment 4 image will be used
2. A requirements.txt in the referenced in the Workspace will be installed
3. my-library package will be installed with version 6.1
4. A wheel in Workspace will be installed

```
None
environment_version: '4'
dependencies:
  - --index-url https://pypi.org/simple
  - -r "/Workspace/Shared/requirements.txt"
  - my-library==6.1
  - /Workspace/Shared/Path/To/simplejson-3.19.3-py3-none-any.whl
```

Axolotl Example

This is the dependency file for the Axolotl training example

```
None
version: "4"
```

```
dependencies:
  # Standard build tools and MLflow
  - packaging
  - setuptools
  - wheel
  - ninja
  - mlflow>=3.6

  # Axolotl with flash-attn and deepspeed (no build isolation)
  - --no-build-isolation
  - axolotl[flash-attn,deepspeed]

  # Flash attention wheel (no deps - specific CUDA/torch version)
  - --no-deps
  -
https://github.com/Dao-AILab/flash-attention/releases/download/v2.7.4.post1/fla
sh_attn-2.7.4.post1+cu12torch2.6cxx11abiFALSE-cp312-cp312-linux_x86_64.whl
```

FAQs

Q: What content is uploaded to the remote GPU instances when I initiate a job from my local machine?

The job's access to files is determined by the workspace where it was launched, which is defined by the profile set in `export DATABRICKS_CONFIG_PROFILE=dev`.

Additionally, if the `code_source` is specified in the workload YAML file, the job can access the content of the linked repository folder. This content is available on the GPU worker at the path `$HOME/<repo_name>`. Note that this remote path may differ from your local repository path. For example, if the local path is `$HOME/work/ml-misc`, the corresponding remote path on the GPU worker would be `$HOME/ml-misc`.

```yaml
code_source:
  type: snapshot
  snapshot:
    # change this to your local repo path
    repo_path: $HOME/work/ml-misc
```

Q: Where should I save my training results, like model checkpoints?

We advise storing your training results within a Unity Catalog Volume. You can reference this volume just as you would a local file path.

Example:

```
output_dir:
/Volumes/share/ml-misc/air-examples/sgcli/axolotl/checkpoints
```