Fine-tune a model with AI Toolkit for VS Code

Article • 05/21/2024

The AI Toolkit for VS Code (AI Toolkit) is a VS Code extension that enables you to download, test, fine-tune, and deploy AI models with your apps or the cloud. For more information, see the AI Toolkit overview.

In this article, you'll learn how to:

✓ Set up a local environment to fine-tune.
✓ Execute a fine-tuning job.

Prerequisites

- Completed Get started with AI Toolkit for Visual Studio Code.
- If you're using a *Windows* computer to fine-tune, install **Windows Subsystem for Linux (WSL)**. See How to install Linux on Windows with WSL to get WSL and a default Linux distribution installed. WSL Ubuntu distribution 18.04 or greater must be installed and set to be the default distribution prior to using AI Toolkit for VS Code. Learn how to change the default distribution.
- If you're using a *Linux* computer, it should be an Ubuntu distribution 18.04 or greater.
- During preview, the AI Toolkit for VS Code only supports NVIDIA GPUs for fine-tuning.

Tip

Ensure you have the latest NVIDIA drivers installed on your computer. If you are given a choice between *Game Ready Driver* or *Studio Driver*, download the **Studio Driver**.

You'll need to know the model of your GPU to download the correct drivers. To find out which GPU you have, see How to check your GPU and why it matters.

Environment set up

To check whether you have all the necessary prerequisites to run fine-tuning jobs on your local device or cloud VM, open the command palette (Shift+Control+P) and search

AI Toolkit: Validate Environment prerequisites.

If your local device passes the validation checks, the **Setup WSL Environment** button will be enabled for you to select. This will install all the dependencies required to run fine-tuning jobs.

Cloud VM

If your local computer does not have an Nvidia GPU device, it is possible to fine-tune on a cloud VM - both Windows and Linux - with an Nvidia GPU (if you have quota). In Azure, you can fine-tune with the following VM series:

- NCasT4_v3-series
- NC A100 v4-series
- ND A100 v4-series
- NCads H100 v5-series
- NCv3-series
- NVadsA10 v5-series

Tip

VS Code allows you to remote into your cloud VM. If you're unfamiliar with this feature, Read the Remote development over SSH tutorial

Fine-tune

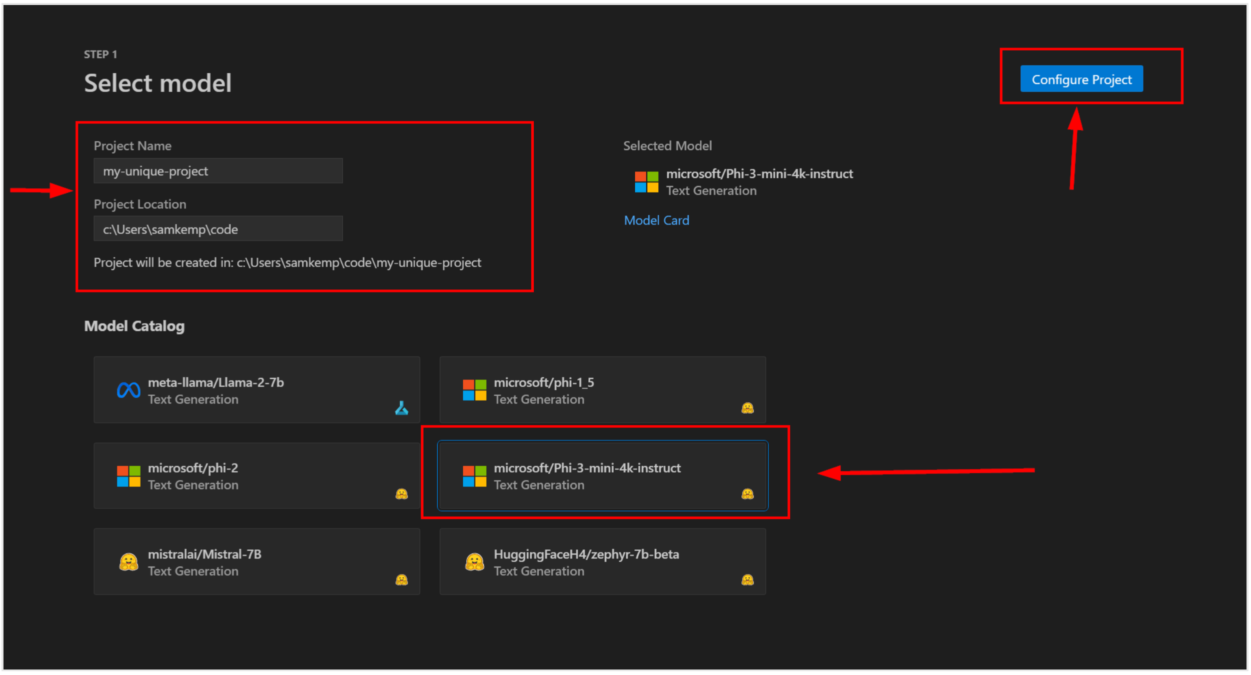
The AI Toolkit uses a method called *QLoRA*, which combines quantization and low-rank adaptation (LoRA) to fine-tune models with your own data. Learn more about QLoRA at QLoRA: Efficient Finetuning of Quantized LLMs .

Step 1: Configure project

To start a new fine-tuning session using QLoRA, select the **Model Fine-tuning** item in AI Toolkit.

Start by entering a unique **Project Name** and a **Project Location**. A new folder with the specified project name will be created in the location you selected to store the project files.

Next, select a model - for example, **Phi-3-mini-4k-instruct** - from the **Model Catalog** and then select **Configure Project**:


STEP 1
Select model

Project Name
my-unique-project
Project Location
c:\Users\samkemp\code
Project will be created in: c:\Users\samkemp\code\my-unique-project

Selected Model
microsoft/Phi-3-mini-4k-instruct
Text Generation
Model Card

Configure Project

Model Catalog

meta-llama/Llama-2-7b
Text Generation

microsoft/phi-1_5
Text Generation

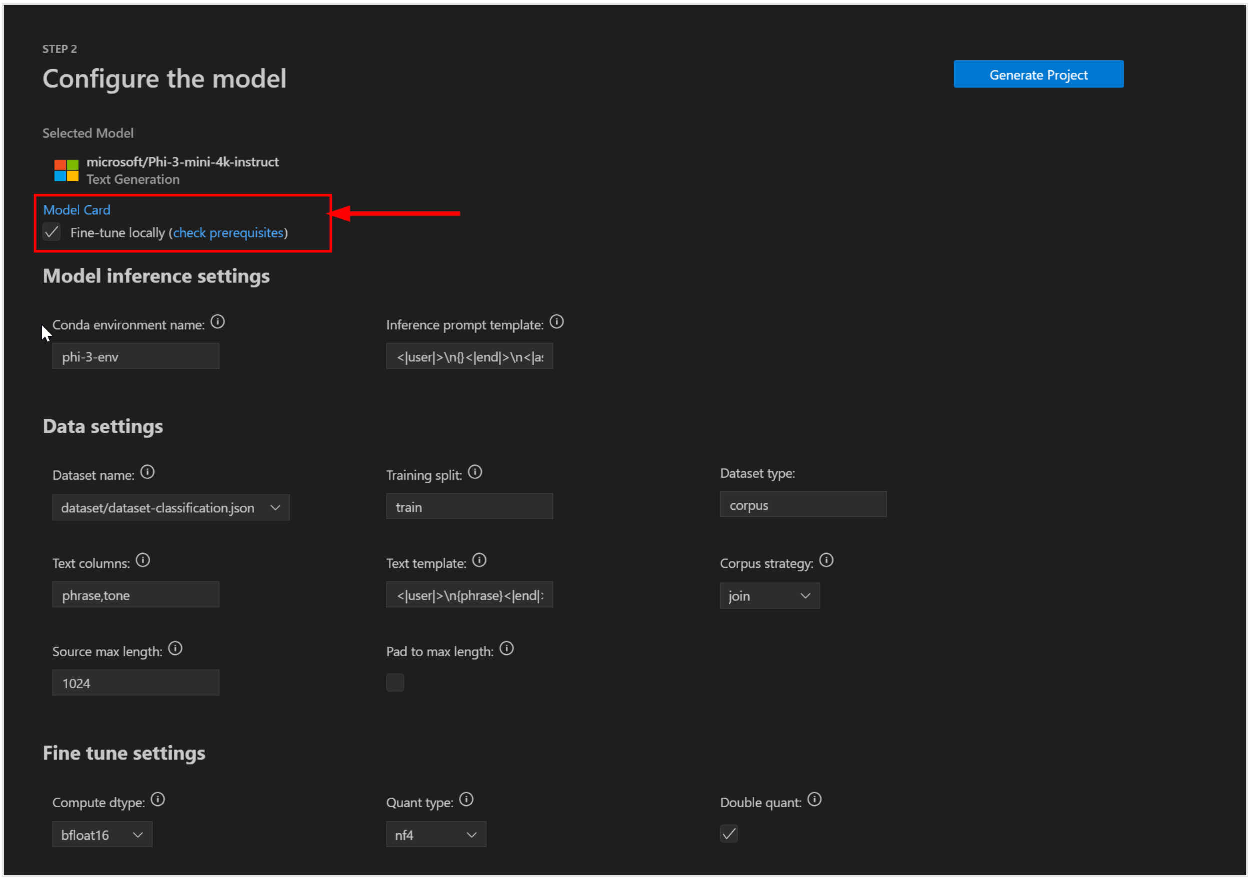
microsoft/phi-2
Text Generation

microsoft/Phi-3-mini-4k-instruct
Text Generation

mistralai/Mistral-7B
Text Generation

HuggingFaceH4/zephyr-7b-beta
Text Generation

You'll then be prompted to configure your fine-tuning project settings. Ensure the **Fine-tune locally** checkbox is ticked (in the future the VS Code extension will allow you to offload fine-tuning to the cloud):


STEP 2
Configure the model

Generate Project

Selected Model
microsoft/Phi-3-mini-4k-instruct
Text Generation
Model Card
✓ Fine-tune locally (check prerequisites)

Model inference settings

Conda environment name: phi-3-env
Inference prompt template: <|user|>\n{}<|end|>\n<|a:

Data settings

Dataset name: dataset/dataset-classification.json
Training split: train
Dataset type: corpus
Text columns: phrase,tone
Text template: <|user|>\n{phrase}<|end|:
Corpus strategy: join
Source max length: 1024
Pad to max length:

Fine tune settings

Compute dtype: bfloat16
Quant type: nf4
Double quant: ✓

Model inference settings

There are two settings available in the **Model inference** section:

Expand table

Setting	Description
Conda environment name	The name of the conda environment to be activated and used for the fine-tuning process. **This name must be unique in your conda installation.**
Inference prompt template	Prompt template to be used at inference time. Make sure this matches the fine-tuned version.

Data settings

The following settings are available in the **Data** section to configure the dataset information:

Expand table

Setting	Description
Dataset name	The name of the dataset to be used for fine-tuning the model.
Training split	The training split name for your dataset.
Dataset type	The type of dataset to be used.
Text columns	The names of the columns in the dataset to populate the training prompt.
Text template	The prompt template to be used to fine tune the model. This uses replacement tokens from the **Text columns**.
Corpus strategy	Indicates if you want to **join** the samples or process them **line by line**.
Source max length	The maximum number of tokens per training sample.
Pad to max length	Add a PAD token to the training sample until the max number of tokens.

Fine-tune settings

The following settings are available in the **Fine tune** section to further configure the fine-tuning process:

Expand table

Settings	Data type	Default value	Description
Compute Dtype	String	bfloat16	The data type for model weights and adapter weights. For a 4bit quantized model, it's also the computation data type for the quantized modules. Valid values: bfloat16, float16, or float32.
Quant type	String	nf4	The quantization data type to use. Valid values: fp4 or nf4.
Double quant	Boolean	yes	Whether to use nested quantization where the quantization constants from the first quantization are quantized again.
Lora r	Integer	64	The Lora attention dimension.
Lora alpha	Float	16	The alpha parameter for Lora scaling.
Lora dropout	Float	0.1	The dropout probability for Lora layers.
Eval dataset size	Float	1024	The size of the validation dataset.
Seed	Integer	0	A random seed for initialization.
Data Seed	Integer	42	A random seed to be used with data samplers.
Per device train batch size	Integer	1	The batch size per GPU for training.
Per device eval batch size	Integer	1	The batch size per GPU for evaluation.
Gradient accumulation steps	Integer	4	The number of updates steps to accumulate the gradients for, before performing a backward/update pass.
Enable Gradient checkpoint	Boolean	yes	Use gradient checkpointing. The is recommended to save memory.
Learning rate	Float	0.0002	The initial learning rate for AdamW.
Max steps	Integer	-1	If set to a positive number, the total number of training steps to perform. This overrides num_train_epochs. In case of using a finite iterable dataset, the training may stop before reaching the set number of steps when all data is exhausted.

Step 2: Generate project

After all the parameters are set, click **Generate Project**. This will perform the following actions:

- Initiate the model download.
- Install all prerequisites and dependencies.
- Create VS Code workspace.

When the model is downloaded and the environment is ready, you can launch the project from AI Toolkit by selecting **Relaunch Window in Workspace** on the **Step 3 - Generating project** page. This will launch a new instance of VS Code connected to your environment.

ⓘ Note

You may be prompted to install additional extensions such as Prompt flow for VS Code. For an optimal fine-tuning experience, install them to proceed.

The relaunched window will have in its workspace the following folders:

Expand table

Folder Name	Description
dataset	This folder contains the dataset for the *template* (`dataset-classification.json` - a JSON lines file containing phrases and tones). If you set your project to use a local file or Hugging Face dataset, you can ignore this folder.
finetuning	The Olive configuration files to execute the fine-tuning job. Olive is an easy-to-use hardware-aware model optimization tool that composes industry-leading techniques across model compression, optimization, and compilation. Given a model and targeted hardware, Olive composes the best suitable optimization techniques to output the most efficient model(s) for inferring on cloud or edge, while taking a set of constraints such as accuracy and latency into consideration.
inference	Code samples for inferencing with a fine-tuned model.
infra	For finetuning and inference using Azure Container App Service (coming soon). This folder contains the Bicep and configuration files to provision the Azure Container App Service.
setup	Files used to set up the conda environment. For example, the pip requirements.

Step 3: Execute fine-tuning job

You can now fine-tune the model using:

Bash

```bash
# replace {conda-env-name} with the name of the environment you set
conda activate {conda-env-name}
python finetuning/invoke_olive.py
```

Important

The time it takes to fine-tune will be dependent on the GPU type, the number of GPUs, the number of steps, and number of epochs. This can be time-consuming (for example, it can take several *hours*).

If you only want to do a quick test, consider reducing the number of *maximum steps* in your `olive-config.json` file. Checkpointing is used and therefore the next fine-tune run will continue from the last checkpoint.

Checkpoints and final model will be saved in `models` folder of your project.

Step 4: Integrate fine-tuned model into your app

Next run inferencing with the fine-tuned model through chats in a `console`, `web browser` or `prompt flow`.

Bash

```bash
cd inference

# Console interface.
python console_chat.py

# Web browser interface allows to adjust a few parameters like max new token
length, temperature and so on.
# User has to manually open the link (e.g. http://127.0.0.1:7860) in a
browser after gradio initiates the connections.
python gradio_chat.py
```

Tip

Instructions are also available in the `README.md` page, which can be found in the project folder.

See Also

- AI Toolkit overview
- Get started with AI Toolkit for Visual Studio Code
- Model fine-tuning concepts