The AI Toolkit for Visual Studio Code (VS Code) was previously known as Windows AI Studio. The extension has been renamed to reflect the focus on enabling AI development in VS Code across platforms.

The AI Toolkit for VS Code (AI Toolkit) is a VS Code extension that enables you to:

Download and run AI models locally. The AI Toolkit provides out-of-the-box access to highly optimized models for the following platforms and hardware:

- Windows 11 running with DirectML acceleration
- Windows 11 running directly on the CPU
- Linux with NVIDIA acceleration
- Linux running directly on the CPU
- Test models in an intuitive playground or in your application with a REST API.

Fine-tune your AI model - locally or in the cloud (on a virtual machine) - to create new skills, improve reliability of responses, set the tone and format of the response. The AI Toolkit provides a guided walkthrough to fine-tune popular small-language models (SLMs) - like Phi-3 and Mistral.

Deploy your AI feature either to the cloud or with an application that runs on a device.

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:

- Install the AI Toolkit for VS Code
- Download a model from the catalog
- Run the model locally using the playground
- Integrate an AI model into your application using REST or the ONNX Runtime

Prerequisites

VS Code must be installed. For more information, see Download VS Code and Getting started with VS Code.

Install

The AI Toolkit is available in the Visual Studio Marketplace and can be installed like any other VS Code extension. If you're unfamiliar with installing VS Code extensions, follow these steps:

- In the Activity Bar in VS Code select Extensions
- In the Extensions Search bar type "AI Toolkit"

- Select the "AI Toolkit for Visual Studio code"
- Select Install

Once the extension has been installed, you'll see the AI Toolkit icon appear in your Activity Bar.

Download a model from the catalog

The primary sidebar of the AI Toolkit is organized into Models and Resources. The Playground and Fine-tuning features are available in the Resources section.

You'll notice that the model cards show the model size, the platform and accelerator type (CPU, GPU). For optimized performance on Windows devices that have at least one GPU, select model versions that only target Windows. This ensures you have a model optimized for the DirectML accelerator. The model names are in the format of {model_name}-{accelerator}-{quantization}-{format}.

To check whether you have a GPU on your Windows device, open Task Manager and then select the Performance tab. If you have GPU(s), they will be listed under names like "GPU 0" or "GPU 1".

Next, download the following model depending on the availability of a GPU on your device.

Run the model in the playground

Once your model has downloaded, select Load in Playground on the model card in the catalog:

In the chat interface of the playground enter the message followed by the Enter key:

You should see the model response streamed back to you

NOTE: If you do not have a GPU available on your device but you selected the Phi-3-mini-4k-directml-int4-awq-block-128-onnx model, the model response will be very slow. You should instead download the CPU optimized version: Phi-3-mini-4k-cpu-int4-rtn-block-32-acc-level-4-onnx.

It is also possible to change:

Context Instructions: Help the model understand the bigger picture of your request. This could be background information, examples/demonstrations of what you want or explaining the purpose of your task.

Inference parameters:

- Maximum response length: The maximum number of tokens the model will return.
- Temperature: Model temperature is a parameter that controls how random a language model's output is. A higher temperature means the model takes more risks, giving you a diverse mix of words. On the other hand, a lower temperature makes the model play it safe, sticking to more focused and predictable responses.
- Top P: Also known as nucleus sampling, is a setting that controls how many possible words or phrases the language model considers when predicting the next word

- Frequency penalty: This parameter influences how often the model repeats words or phrases in its output. The higher the value (closer to 1.0) encourages the model to avoid repeating words or phrases.
- Presence penalty: This parameter is used in generative AI models to encourage diversity and specificity in the generated text. A higher value (closer to 1.0) encourages the model to include more novel and diverse tokens. A lower value is more likely for the model to generate common or cliche phrases.

Integrate an AI model into your application

There are two options to integrate the model into your application:

- The AI Toolkit comes with a local REST API web server that uses the OpenAI chat completions format. This enables you to test your application locally - using the endpoint http://127.0.0.1:5272/v1/chat/completions - without having to rely on a cloud AI model service. Use this option if you intend to switch to a cloud endpoint in production. You can use OpenAI client libraries to connect to the web server.
- Using the ONNX Runtime. Use this option if you intend to ship the model with your application with inferencing on device.

Local REST API web server

The local REST API web server allows you to build-and-test your application locally without having to rely on a cloud AI model service. You can interact with the web server using REST, or with an OpenAI client library

Python code:

pip install openai

from openai import OpenAI

```
client = OpenAI(
    base_url="http://127.0.0.1:5272/v1/",
    api_key="x" # required by API but not used
)

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "what is the golden ratio?",
        }
```

```
    ],
    model="Phi-3-mini-4k-directml-int4-awq-block-128-onnx",
)

print(chat_completion.choices[0].message.content)
```

ONNX Runtime

The ONNX Runtime Generate API provides the generative AI loop for ONNX models, including inference with ONNX Runtime, logits processing, search and sampling, and KV cache management. You can call a high level generate() method, or run each iteration of the model in a loop, generating one token at a time, and optionally updating generation parameters inside the loop.

It has support for greedy/beam search and TopP, TopK sampling to generate token sequences and built-in logits processing like repetition penalties. The following code is an example of how you can leverage the ONNX runtime in your applications.

pip install numpy

Python Code:

```
# app.py
import onnxruntime_genai as og
import argparse

def main(args):
    print("Loading model...")
    model = og.Model(f'{args.model}')
    print("Model loaded")
    tokenizer = og.Tokenizer(model)
    tokenizer_stream = tokenizer.create_stream()
    search_options = {
        'max_length': 2048
    }

    chat_template = '<|user|>\n{input} <|end|>\n<|assistant|>'
```

```
    # Keep asking for input prompts in a loop
    while True:
        text = input("Input: ")

        # If there is a chat template, use it
        prompt = f'{chat_template.format(input=text)}'

        input_tokens = tokenizer.encode(prompt)

        params = og.GeneratorParams(model)
        params.set_search_options(**search_options)
        params.input_ids = input_tokens

        generator = og.Generator(model, params)
        print("\nOutput: ", end='', flush=True)
        while not generator.is_done():
            generator.compute_logits()
            generator.generate_next_token()
            new_token = generator.get_next_tokens()[0]
            print(tokenizer_stream.decode(new_token), end='', flush=True)

        print()
        print()

        # Delete the generator to free the captured graph for the next generator, if graph capture is enabled
        del generator


if __name__ == "__main__":
```

```
    parser = argparse.ArgumentParser()
    parser.add_argument('-m', '--model', type=str, required=True, help='Onnx model folder path (must contain config.json and model.onnx)')
    args = parser.parse_args()
    main(args)
```

To run the Python app use the following code:

```
python app.py --model ~/.aitk/models/{path_to_folder_containing_onnx_file}
```

Note

The AI Toolkit caches model downloads into a hidden folder named .aitk in your user directory - you'll need to update the path used for the --model parameter to the location of the folder containing the ONNX model file. For example ~/.aitk/models/microsoft/Phi-3-mini-4k-instruct-onnx/directml/Phi-3-mini-4k-directml-int4-awq-block-128-onnx/