# My New Copilot+ PC

I first saw the Copilot+ PC device on stage at the Microsoft BUILD conference in May 2024, during a demonstration of how its NPU could dramatically speed up AI tasks running locally, and ever since, I’d wanted one. I was lucky to get one of the few Qualcomm Windows on Snapdragon X Elite Dev Kits before they halted production. Although Qualcomm-powered ARM Copilot+ PC laptops have been shipping since mid-June, my Dev Kit didn’t arrive until mid-October, about three months later, and I was eager to see what I could do with its Hexagon NPU. Here’s how it’s going…

## Windows on ARM

Thus far, as far as a new computer goes, I’m absolutely thrilled. The Dev Kit immediately became my main computer, and I won’t look back. The VPN I use (OpenVPN) as well as a couple of other programs are not yet available natively for Windows on ARM yet, but they install and run absolutely fine under x64 emulation, and you wouldn’t know they’re x64 apps unless you peek in Task Manager. It does everything my Intel computers do, and I’ve not had a single issue with it or any software running on it. It boots almost instantly, it’s blazing fast, cool, and quiet. I can even run Visual Studio in a Teams meeting without everything slowing to a crawl, something I was never quite able to accomplish on Intel.

I’d been toying with the idea of getting an ARM machine as my next computer for a couple of years. I knew that Windows for ARM64 with its impressive emulation of x86 and x64 had been around for a few years and rumors were that it was getting quite stable. I also knew that there were native ARM64 versions of Visual Studio 2022 and VS Code and that the Office applications were being converted to native ARM64. Any bits of Office that aren’t yet converted run in emulation and communication between the two platforms are handled automatically. This hybrid technology is called ARM64EC and is available to all C++ application developers. When Copilot+ PCs were announced, it seemed that the momentum was there to make Windows on ARM mainstream. The only complaints I’d heard while waiting for my machine to arrive were that a lot of games and some other software, (e.g., third-party VPNs) don’t run on them yet and won’t until the vendors build native ARM64 versions. I’m not a gamer, but friends with their own ARM Copilot+ PCs tell me the gaming experience isn’t bad. Plus, I still have my i7 laptop to use if I ever need it. So far, I haven’t.

## Copilot+ PC Features

As announced at BUILD, Microsoft is planning on adding an AI stack to the Windows OS, both for ARM and Intel. In <https://learn.microsoft.com/en-us/windows/ai/npu-devices/#how-to-access-the-npu-on-a-copilot-pc>, Microsoft lists “Unique AI features supported by Copilot+ PCs with an NPU” announced at the conference, which are meant to be a starting point for capabilities to be built into Windows:

* **Windows Studio Effects**: A set of audio and video NPU-accelerated AI effects from Microsoft including Creative Filter, Background Blur, Eye Contact, Auto Framing, Voice Focus. Developers can also add toggles to their app for system level controls.
* **Recall:** The AI-supported UserActivity API that enables users to search for past interactions using natural language and pick up where they left off.
* **Phi Silica:** The Phi Small Language Model (SLM) that enables your app to connect with the on-device model to perform natural language processing tasks (chat, math, code, reasoning) using an upcoming release of the Windows App SDK.
* **Text Recognition:** The Optical Character Recognition (OCR) API that enables the extraction of text from images and documents.
* **Cocreator with Paint:** A new feature in Microsoft Paint that transforms images into AI Art.
* **Super Resolution:** An industry-leading AI technology that uses the NPU to make games run faster and look better.

I was eager to dive in. Because the Dev Kit computer isn’t a laptop, I plugged in an HD web cam I’d been using on a Linux machine sitting on my desk. The camera doesn’t support IR, so I can’t use it for Windows Hello and, it turns out, I can’t use it for Windows Studio Effects either. It’s unclear what the requirements actually are. My friends with Surface Copilot+ PCs say they like features of Windows Studio Effects, such as having the OS blur the camera background for meetings, as it does a better job than Teams and doesn’t use as much power. But overall, their reviews of Windows Studio Effects are, “meh… It’s nice.”

As you probably heard, Recall has been delayed for a variety of reasons, including privacy issues. It's slated to be rolled out December 2024, and requires you to be on the Windows Insider Program, but we’ll see (this was written in November 2024). It sounds interesting, but I’m not all that excited about it, to be honest, and this isn’t the first new release date it’s had.

One thing I was REALLY excited about is Phi Silica. I do a lot of AI work and being able to tap into the GPU and NPU on this machine with small language models like the Phi-3 family, provided by the OS, would be game changing. Unfortunately, this doesn’t exist yet either. Programmatic access to the local Phi models, when it does arrive, is planned to be part of the Windows App SDK. This is the SDK that includes things like WinUI3 (successor to UWP), power management, app notifications, etc. Basically, it provides high-level access to Windows-specific things so you don’t have to resort to Win32. This makes sense because Windows will host the AI stack and it’s not a cross-platform feature. All a developer will have to do is make high-level calls to use the models on Windows. These features were supposed to be part of the 1.6 SDK release in September 2024, but have been bumped to at least v1.7, which is not yet available, even in preview, as I write this.

Text Recognition is also of interest to me, but it’s also supposed to be released as part of the SDK, and I haven’t had time to even look to see if it’s there yet. To be honest, I have existing ways of doing OCR and it’s not high on my priority list.

This brings us to Cocreator with Paint. This actually works! I brought up Paint and it downloaded a model automatically the first time I used Cocreator. I can create images locally by giving it a prompt, and optionally by asking it to base the image on something I’ve already drawn in Paint. It uses my NPU, which is exciting because it’s the first time I’ve seen the NPU do anything at all. Unfortunately, it’s not nearly as good as other services I use online such as Leonardo.ai, Ideogram, or Midjourney. Still, it’s fun, and it can create images based on what I’ve already drawn, which most online services don’t. I’ve used it to create images for some upcoming AI presentations I’m working on and it’s not bad for a v1 product that runs locally.

As a non-gamer, I haven’t tried Automatic Super Resolution (based on DirectX) to speed up frame rates on video games. Based on community feedback so far, there’s not a lot of excitement yet.

A disappointing one out of six was successful so far using Windows-provided AI features. However, it’s still early days. I’m reminded that it was two years after the Wright brothers’ first flight before they could take off, fly in a circle, and land.

## AI Development on ARM

Phi Silica was a disappointment, but I think it will be a reality soon. Until then of course, I’m a developer and I’m most excited about doing “real” development. I mainly use C# and TSQL, but I also write a little C++ for an open-source project named Photino that’s a .NET-powered Electron look-alike. For C++ developers, there are some things to celebrate. Many legacy projects can be cross compiled for ARM32 or ARM64 (on either hardware) just by adding a new configuration to the Configuration Manager in Visual Studio and recompiling. If you can’t convert all your code and assets at once, you can select ARM64EC just as easily and port your application over time. In addition, Qualcomm has a lot of drivers and sample code available, if you’re into that sort of thing. They have a fairly active Slack workspace and are very responsive. However, at this point, if you’re not a C++ or Python developer working on Linux, there’s not a whole lot you can accomplish on the CPU or GPU.

I spent several weeks configuring Windows Subsystem for Linux (WSL), installing and configuring various versions of Python for x86, x64, and ARM64, downloading SDKs and drivers from Qualcomm, running through tutorials, and troubleshooting with support on Slack. I did end up getting one small image manipulation model to run on my NPU, but I was unable to get any transformer models to run on either the GPU or NPU. This kind of work requires a FAST internet connection, a LOT of hard disk space (the built-in 1/2TB drive is just big enough to convert one small language model like llama-v3-8b-instruct, or Phi-3-medium-128k-instruct, and a lot of patience, because compiling and quantizing models can take hours.

Again, there was a bit of a disappointment. Right now, this type of development is very low level, very finnicky, and you’ll often find yourself waiting for a bug resolution or for something that hasn’t been released yet. The things I was able to accomplish only ran in Linux with Python.

## ONNX to the Rescue (Sort of)

In my experience as a .NET developer, living and breathing AI development on this device, the best, and easiest-to-achieve results have come when I use ONNX. The ONNX model format and ONNX runtime are an attempt to standardize access to local AI models, improve their performance, and make them easier to run on a variety of hardware, including CPUs, GPUs, NPUs, and even within browsers using WebGPU. The easiest path is to find a model, already converted to ONNX format and tuned for the hardware you want to run it on, download it to your machine, and then use the ONNX runtime along with an Execution Provider (EP) created for your hardware. By default, ONNX uses CPU if no provider is mentioned or as a fallback if the specialized provider can’t be used, like, for example, if you run the model on a machine that doesn’t have an NPU or supported GPU.

ONNX can make development easier on several levels that are very important if you want to run models locally.

ONNX can make development easier on several levels that are very important if you want to run models locally. Language Transformer models in particular (there are many other types of models), start out life pretty huge. Even relatively small models designed to run locally, like Phi3.5-mini, include several GBs of data, stored as 32bit floating point numbers. Few models require that amount of precision. For use on GPUs, models are often “quantized” down to 16bit floating-point numbers, cutting the model size to about half (models aren’t all data). CPUs tend to work best with 8bit or even 4bit integers, making the resulting models even smaller. There are even 1bit models that show a lot of promise. Although there may be some loss of capability from the quantizing process, there are several techniques that result in negligible differences in capability. In addition to taking up less space and being faster to load, smaller models also run faster at runtime. An ONNX format model quantized for CPU tends to run multiple times faster than the original non-quantized model. In addition, converting to ONNX, even while retaining the 32bit floating point, will make the model smaller and faster due to things like “fusing” operations and activations. Other techniques such as “distillation” can also be applied to reduce model size and improve performance.

Learning to optimize models well is a large topic, even with tools like Olive that significantly streamline the process of making ONNX models. Luckily, others have done a lot of this work for you and in many cases, you can simply download an ONNX model tuned for CPU and just use it. The Phi-3.5-mini-instruct-onnx model for CPU and mobile is only about 2.6GB and it performs about as well as ChatGPT 3.5, in my experience, and you can download it, ready to go, here: <https://huggingface.co/microsoft/Phi-3.5-mini-instruct-onnx/tree/main/cpu_and_mobile/cpu-int4-awq-block-128-acc-level-4>.

Next, create a new .NET application. A Console app will do. And add the Microsoft.ML.Runtime.OnnxRuntimeGenAI NuGet package. I’m using version 0.5.0 for this article. Replace the contents of the Program.cs file with the code shown in **Listing 1**. Modify the path to the folder containing your copy of the Phi model that you downloaded above and run the code.

You should see output similar to:

Model loading took 2685 ms

Taos, New Mexico, is a picturesque town in the

Sangre de Cristo Mountains, known for its vibrant

arts scene, historic adobe buildings, and annual

art fairs. It's a hub for Native American and

Hispanic cultures, offering a unique blend of

traditions, art, and music. The town also

provides stunning natural landscapes for hiking,

skiing, and stargazing.

You may see some errors in the output. They appear to be caused by unresolved issues in version 0.5.0 and can be safely ignored. They should be resolved in an upcoming release.

Alternately, you can use Semantic Kernel to make use of local ONNX models. It will also support providers for GPU and NPU when they become available.

## Summary

Overall, I’m absolutely thrilled by my new Snapdragon-powered Copilot+ PC ARM computer. These are great machines, and I look forward to the next generation. When it comes to the actual Copilot+ PC features, Windows support for AI, and targeting the GPU and NPU hardware, there just isn’t much “there” there. At least not yet, but there’s progress every day and I’ll be ready as each new feature arrives. As a .NET developer, I could already find, download, and run ONNX models tuned for CPU on my Intel laptop. So far, the new hardware is just a bit faster at the same approach, but I’m looking forward to being able to use this new hardware and using AI offline, and for free, at blazing speeds.

Sponsored Sidebar

Adding Copilots to Your Apps

**The future is here now** and you don’t want to get left behind. Unlock the **true potential** of your software applications by adding **Copilots**.

**CODE Consulting** can assess your applications and provide you with a roadmap for adding Copilot features and optionally assist you in adding them to your applications.

**Reach out to us** today to get your application assessment scheduled: [www.codemag.com/ai](http://www.codemag.com/ai)

Listing 1: Complete source code for Phi3.5 ONNX sample.

using Microsoft.ML.OnnxRuntimeGenAI;

using System.Diagnostics;

using System.Text;

namespace MyNewCopilot\_PC;

internal class Program

{

//Download from <https://huggingface.co/microsoft/>

//Phi-3.5-mini-instruct-onnx/tree/main

  private static readonly string modelDir =

    @"<your localpath>\cpu\_and\_mobile\

cpu-int4-awq-block-128-acc-level-4";

    static async Task Main(string[] args)

    {

      Console.WriteLine($"Loading model: {modelDir}");

      var sw = Stopwatch.StartNew();

      using var model = new Model(modelDir);

      using var tokenizer = new Tokenizer(model);

      sw.Stop();

      Console.WriteLine($"Model loading took {sw.ElapsedMilliseconds} ms");

      var systemPrompt = "You are a helpful assistant.";

      var userPrompt = "Tell me about Taos New Mexico. Be brief.";

      var prompt = $@"<|system|>{systemPrompt}<|end|><|user|>{userPrompt}<|end|>

<|assistant|>";

      await foreach (var part in InferStreaming(prompt, model,

tokenizer))

      Console.Write(part);

 }

  public static async IAsyncEnumerable<string>

InferStreaming(string prompt, Model model, Tokenizer tokenizer)

  {

    using var generatorParams = new GeneratorParams(model);

    using var sequences = tokenizer.Encode(prompt);

    generatorParams.SetSearchOption("max\_length", 2048);

    //generatorParams.SetSearchOption("top\_p", 0.5);

    //generatorParams.SetSearchOption("top\_k", 1);

    //generatorParams.SetSearchOption("temperature", 0.8);

    generatorParams.SetInputSequences(sequences);

    generatorParams.TryGraphCaptureWithMaxBatchSize(1);

    using var tokenizerStream = tokenizer.CreateStream();

    using var generator = new Generator(model, generatorParams);

    StringBuilder stringBuilder = new();

    while (!generator.IsDone())

    {

      string part;

      await Task.Delay(10).ConfigureAwait(false);

       generator.ComputeLogits();

       generator.GenerateNextToken();

      part = tokenizerStream.Decode(generator.GetSequence(0)[^1]);

      stringBuilder.Append(part);

      if (stringBuilder.ToString().Contains("<|end|>")

       || stringBuilder.ToString().Contains("<|user|>")

       || stringBuilder.ToString().Contains("<|system|>"))

          break;

    if (!string.IsNullOrWhiteSpace(part))

        yield return part;

    }

  }

}