Ryzen AI

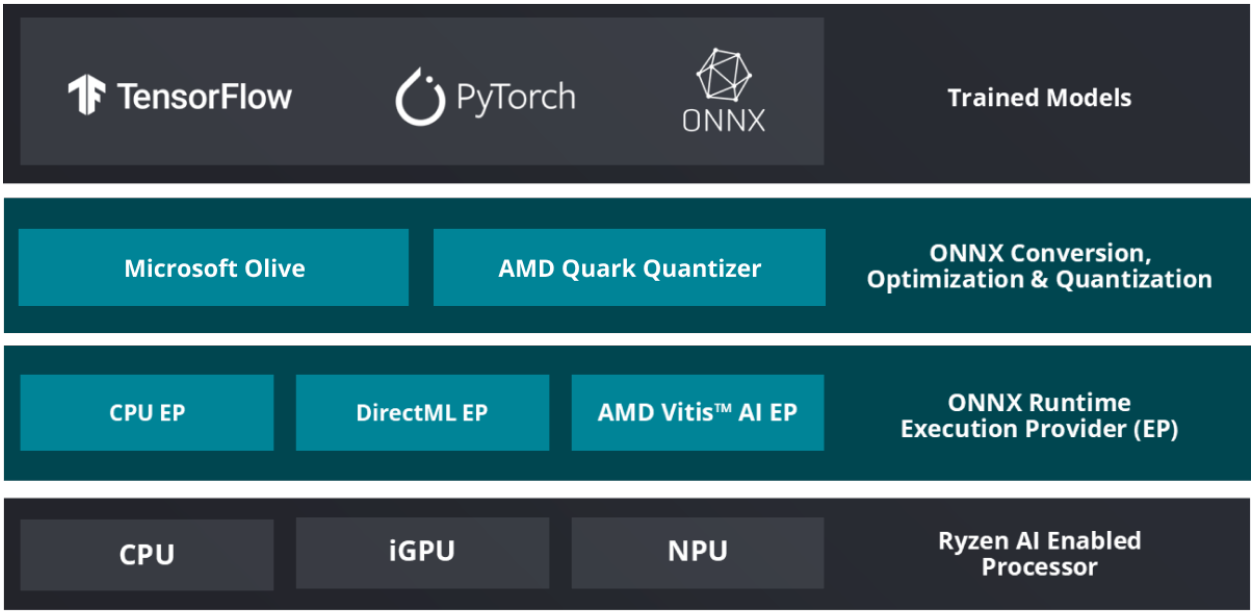
Release 1.5

AMD

Oct 08, 2025

CONTENTS

AMD Ryzen™ AI Software includes the tools and runtime libraries for optimizing and deploying AI inference on AMD Ryzen™ AI powered PCs. Ryzen AI software enables applications to run on the neural processing unit (NPU) built in the AMD XDNA™ architecture, as well as on the integrated GPU. This allows developers to build and deploy models trained in PyTorch or TensorFlow and run them directly on laptops powered by Ryzen AI using ONNX Runtime and the Vitis™ AI Execution Provider (EP).



CHAPTER

ONE

QUICK START

- *Supported Configurations*
- *Installation Instructions*
- *Examples, Demos, Tutorials*

CHAPTER

TWO

DEVELOPMENT FLOW OVERVIEW

The Ryzen AI development flow does not require any modifications to the existing model training processes and methods. The pre-trained model can be used as the starting point of the Ryzen AI flow.

2.1 Quantization

Quantization involves converting the AI model's parameters from floating-point to lower-precision representations, such as bfloat16 floating-point or 8-bit integer. Quantized models are more power-efficient, utilize less memory, and offer better performance.

AMD Quark is a comprehensive cross-platform deep learning toolkit designed to simplify and enhance the quantization of deep learning models. Supporting both PyTorch and ONNX models, Quark empowers developers to optimize their models for deployment on a wide range of hardware backends, achieving significant performance gains without compromising accuracy.

For more details, refer to the *Model Quantization* page.

2.2 Compilation and Deployment

The AI model is deployed using the ONNX Runtime with either C++ or Python APIs. The Vitis AI Execution Provider included in the ONNX Runtime intelligently determines what portions of the AI model should run on the NPU, optimizing workloads to ensure optimal performance with lower power consumption.

For more details, refer to the *Model Compilation and Deployment* page.

2.2.1 Release Notes

Supported Configurations

Ryzen AI 1.5 Software supports AMD processors codenamed Phoenix, Hawk Point, Strix, Strix Halo, and Krackan Point. These processors can be found in the following Ryzen series:

- Ryzen 200 Series
- Ryzen 7000 Series, Ryzen PRO 7000 Series
- Ryzen 8000 Series, Ryzen PRO 8000 Series
- Ryzen AI 300 Series, Ryzen AI PRO Series, Ryzen AI Max 300 Series

For a complete list of supported devices, refer to the processor specifications page (look for the "AMD Ryzen AI" column towards the right side of the table, and select "Available" from the pull-down menu).

The rest of this document will refer to Phoenix as PHX, Hawk Point as HPT, Strix and Strix Halo as STX, and Krackan Point as KRK.

Model Compatibility Table

The following table lists which types of models are supported on what hardware platforms.

Model Type	PHX/HPT	STX/KRK
CNN INT8		
CNN BF16		
NLP BF16		
LLM (OGA)		

Version 1.5

- EoU Improvement
  - Application concurrency: improves the resource distribution across applications
  - Model Compilation time: 2x – 8x faster
  - Installation Size: 80% smaller
- Stable Diffusion demo pipelines (preview)
- 4K context length supported (on selected models)
- LLM Context cache support (on selected models)
- Bug fixes
- New LLMs released

  - Qwen/Qwen2.5-1.5B-Instruct
  - Qwen/Qwen2.5-3B-Instruct
  - Qwen/Qwen2.5-7B-Instruct

- Breaking Changes
  - The `%RYZEN_AI_INSTALLATION_PATH%\deployment` folder has been reorganized and flattened. Deployment DLLs are no longer organized in subfolders. If you use application build scripts that pull DLLs from the `deployment` folder, you need to update them based on the new paths. Refer to the *Application Packaging Requirements* section for further details.
  - The `1x4.xclbin` (PHX/HPT) and `AMD_AIE2P_Nx4_Overlay.xclbin` (STX/KRK) NPU binaries are no longer supported and should not be used. You should use the `4x4.xclbin` (PHX/HPT) and `AMD_AIE2P_4x4_Overlay.xclbin` (STX/KRK) NPU binaries instead.
  - The `XLNX_ENABLE_CACHE`, `XLNX_VART_FIRMWARE`, and `XLNX_TARGET_NAME` environment variables are no longer supported and should not be relied upon.
  - Support for VitisAI EP cache encryption is no longer available. To encrypt the compiled models, use the ONNX Runtime *EP Context Cache* feature instead.
  - For INT8 models, the VitisAI EP does not save the compiled model to disk by default. To save the compiled model, use the ONNX Runtime *EP Context Cache* feature or set the *enable_cache_file_io_in_mem* provider option to 0.
  - Generation of the `vitisai_ep_report.json` file is no longer automatic and should be manually enabled. See the *Operator Assignment Report* section for details.
  - Changes to the OGA flow for LLMs:
    * OGA Version is updated to **v0.7.0** (Ryzen AI 1.5) from v0.6.0 (Ryzen AI 1.4).
    * The `hybrid_llm` and `npu_llm` folders are consolidated into a new folder named `LLM`, which contains the `model_benchmark.exe` and `run_model.py` scripts, along with the necessary C++ headers and .lib files to support both the Hybrid LLM and NPU LLM workflows in C++ and Python.
    * For NPU LLM models, the `vaip_llm.json` file is no longer required. As a result, the `vaip_llm.json` path is removed from the `genai_config.json` for all NPU models. Ensure that you re-download the NPU models from Hugging Face when using the Ryzen AI 1.5 installer.

Version 1.4

- New Features:
  - New architecture support for Ryzen AI 300 series processors
  - Unified support for LLMs, INT8, and BF16 models in a single release package
  - Public release for compilation of BF16 CNN and NLP models on Windows
  - Public release of the LLM Hybrid OGA flow
  - LLM building flow for finetuned LLM
  - Support for up to 16 hardware contexts on Ryzen AI 300 series processors
  - Vitis AI EP now supports the ONNX Runtime EP context cache feature (for custom handling of pre-compiled models)
  - Ryzen AI environment variables converted to VitisAI EP session options
  - Improved exception handling and fallback to CPU
- New Hybrid execution mode LLMs:
  - DeepSeek-R1-Distill-Llama-8B
  - DeepSeek-R1-Distill-Qwen-1.5B
  - DeepSeek-R1-Distill-Qwen-7B
  - Gemma2-2B
  - Qwen2-1.5B
  - Qwen2-7B
  - AMD-OLMO-1B-SFT-DPO
  - Mistral-7B-Instruct-v0.1
  - Mistral-7B-Instruct-v0.2
  - Mistral-7B-v0.3
  - Llama3.1-8B-Instruct
  - Codellama-7B-Instruct
- *New BF16 model examples*:
  - Image classification
  - Finetuned DistilBERT for text classification
  - Text embedding model Alibaba-NLP/gte-large-en-v1.5
- New Tools:

  - Lemonade SDK
    * Lemonade Server: A server interface that uses the standard Open AI API, allowing applications in any language to integrate with Lemonade Server for local LLM deployment and compatibility with existing Open AI apps.
    * Lemonade Python API: Offers High-Level API for easy integration of Lemonade LLMs into Python applications and Low-Level API for custom experiments with specific checkpoints, devices, and tools.
    * Lemonade Command Line Interface easily benchmark, measure accuracy, prompt or gather memory usage of your LLM.
  - TurnkeyML – Open-source tool that includes low-code APIs for general ONNX workflows.
  - Digest AI – A Model Ingestion and Analysis Tool in collaboration with the Linux Foundation.
  - GAIA – An open-source application designed for the quick setup and execution of generative AI applications on local PC hardware.
- Quark-torch:
  - Added ROUGE and METEOR evaluation metrics for LLMs
  - Support for evaluating ONNX models exported using OGA
  - Support for offline evaluation (evaluation without generation) for LLMs
  - Support for Hugging Face integration
  - Support for Gemma2 quantization using the OGA flow
  - Support for Llama-3.2 quantization with FP8 (weights, activation, and KV-cache) for the vision and language components
- Quark-onnx:
  - Support compatibility with ONNX Runtime version 1.20.0, and 1.20.1
  - Support for microexponents (MX) data types, including MX4, MX6, and MX9
  - Support for BF16 data type for VAIML
  - Support for excluding pre and post-processing from quantization
  - Support for mixed precision with any data type
  - Support for Quarot rotation R1 algorithm
  - Support for microexponents and microscaling AdaQuant
  - Support for an auto-search algorithm to automatically find the best accuracy quantized model

  - Added tools for evaluating L2, PSNR, VMAF, and cosine

- ONNX Runtime EP:
  - Support for Chinese characters in the `filename/cache_dir/cache_key/xclbin`
  - Support for `int4/uint4` data type
  - Support for configurable failure handling: CPU fallback or exception
  - Update for encrypt/decrypt feature
- Known Issues:
  - Microsoft Windows Insider Program (WIP) users may see warnings or need to restart when running all applications concurrently.
    * NPU driver and workloads will continue to work.
  - Context creation may appear to be limited when some application do not close contexts quickly.

Version 1.3

- New Features:
  - Initial release of the Quark quantizer
  - Support for mixed precision data types
  - Compatibility with Copilot+ applications
- Improved support for *LLMs using OGA*
- New EoU Tools:
  - CNN profiling tool for VAI-ML flow
  - Idle detection and suspension of contexts
  - Rebalance feature for AIE hardware resource optimization
- NPU and Compiler:
  - New Op Support:
    * MAC
    * QResize Bilinear
    * LUT Q-Power
    * Expand
    * Q-Hsoftmax
    * A16 Q-Pad

        * Q-Reduce-Mean along H/W dimension
        * A16 Q-Global-AvgPool
        * A16 Padding with non-zero values
        * A16 Q-Sqrt
        * Support for XINT8/XINT16 MatMul and A16W16/A8W8 Q-MatMul
    - Performance Improvements:
        * Q-Conv, Q-Pool, Q-Add, Q-Mul, Q-InstanceNorm
        * Enhanced QDQ support for a range of operations
        * Enhanced the tiling algorithm
        * Improved graph-level optimization with extra transpose removal
        * Enhanced AT/MT fusion
        * Optimized memory usage and compile time
        * Improved compilation messages
- Quark for PyTorch:
    - Model Support:
        * Examples of LLM PTQ, such as Llama3.2 and Llama3.2-Vision models
        * Example of YOLO-NAS detection model PTQ/QAT
        * Example of SDXL v1.0 with weight INT8 activation INT8
    - PyTorch Quantizer Enhancements:
        * Partial model quantization by user configuration under FX mode
        * Quantization of ConvTranspose2d in Eager Mode and FX mode
        * Advanced Quantization Algorithms with auto-generated configurations
        * Optimized Configuration with DataTypeSpec for ease of use
        * Accelerated in-place replacement under Eager Mode
        * Loading configuration from file of algorithms and pre-optimizations
- Quark for ONNX:
    - New Features:
        * Compatibility with ONNX Runtime version 1.18, 1.19
        * Support for int4, uint4, Microscaling data types
        * Quantization for arbitrary specified operators

        - Quantization type alignment of element-wise operators for mixed precision
        - ONNX graph cleaning
        - Int32 bias quantization
    - ONNX Quantizer Enhancements:
        - Fast fine-tuning support for the MatMul operator, BFP data type, and GPU acceleration
        - Improved ONNX quantization of LLM models
        - Optimized quantization of FP16 models
        - Custom operator compilation process
        - Default parameters for auto mixed precision
        - Optimized Ryzen AI workflow by aligning with hardware constraints of the NPU
- ONNX Runtime EP:
    - Support for ONNX Runtime EP shared libraries
    - Python dependency removal
    - Memory optimization during the compile phase
    - Pattern API enhancement with multiple outputs and commutable arguments support
- Known Issues:
    - Extended compile time for some models with BF16/BFP16 data types
    - LLM models with 4K sequence length may revert to CPU execution
    - Accuracy drop in some Transformer models using BF16/BFP16 data types, requiring Quark intervention

Version 1.2

- New features:
    - Support added for Strix Point NPUs
    - Support added for integrated GPU
    - Smart installer for Ryzen AI 1.2
    - NPU DPM based on power slider
- New model support:
    - LLM flow support for multiple models in both PyTorch and ONNX flow (optimized model support will be released asynchronously)

  - SDXL-T with limited performance optimization
- New EoU tools:
  - AI Analyzer : Analysis and visualization of model compilation and inference profiling
  - Platform/NPU inspection and management tool (xrt-smi)
  - Onnx Benchmarking tool
- New Demos:
  - NPU-GPU multi-model pipeline application demo
- NPU and Compiler
  - New device support: Strix Nx4 and 4x4 Overlay
  - New Op support:
    * InstanceNorm
    * Silu
    * Floating scale quantization operators (INT8, INT16)
  - Support new rounding mode (Round to even)
  - Performance Improvement:
    * Reduced the model compilation time
    * Improved instruction loading
    * Improved synchronization in large overlay
    * Enhanced strided_slice performance
    * Enhanced convolution MT fusion
    * Enhanced convolution AT fusion
    * Enhanced data movement op performance
- ONNX Quantizer updates
  - Improved usability with various features and tools, including weights-only quantization, graph optimization, dynamic shape fixing, and format transformations.
  - Improved the accuracy of quantized models through automatic mixed precision and enhanced AdaRound and AdaQuant techniques.
  - Enhanced support for the BFP data type, including more attributes and shape inference capability.
  - Optimized the NPU workflow by aligning with the hardware constraints of the NPU.
  - Supported compilation for Windows and Linux.

  - Bugfix:
    * Fixed the problem where per-channel quantization is not compatible with onnxruntime 1.17.
    * Fixed the bug of CLE when conv with groups.
    * Fixed the bug of bias correction.
- Pytorch Quantizer updates
  - Tiny value quantization protection.
  - Higher onnx version support in quantized model exporting.
  - Relu6 hardware constrains support.
  - Support of mean operation with keepdim=True.
- Resolved issues:
  - NPU SW stack will fail to initialize when the system is out of memory. This could impact camera functionality when Microsoft Effect Pack is enabled.
  - If Microsoft Effects Pack is overloaded with other 4+ applications that use NPU to do inference, then camera functionality can be impacted. Can be fixed with a reboot. This will be fixed in the next release.

Version 1.1

- New model support:
  - Llama 2 7B with w4abf16 (3-bit and 4-bit) quantization (Beta)
  - Whisper base (EA access)
- New EoU tools:
  - CNN Benchmarking tool on RyzenAI-SW Repo
  - Platform/NPU inspection and management tool

Quantizer

- ONNX Quantizer:
  - Improved usability with various features and tools, including diverse parameter configurations, graph optimization, shape fixing, and format transformations.
  - Improved quantization accuracy through the implementation of experimental algorithmic improvements, including AdaRound and AdaQuant.
  - Optimized the NPU workflow by distinguishing between different targets and aligning with the hardware constraints of the NPU.

  - Introduced new utilities for model conversion.

- PyTorch Quantizer:
  - Mixed data type quantization enhancement and bug fix.
  - Corner bug fixes for add, sub, and conv1d operations.
  - Tool for converting the S8S8 model to the U8S8 model.
  - Tool for converting the customized Q/DQ to onnxruntime contributed Q/DQ with the "microsoft" domain.
  - Tool for fixing a dynamic shapes model to fixed shape model.
- Bug fixes
  - Fix for incorrect logging when simulating the LeakyRelu alpha value.
  - Fix for useless initializers not being cleaned up during optimization.
  - Fix for external data cannot be found when using use_external_data_format.
  - Fix for custom Ops cannot be registered due to GLIBC version mismatch

NPU and Compiler

- New op support:
  - Support Channel-wie Prelu.
  - Gstiling with reverse = false.
- Fixed issues:
  - Fixed Transpose-convolution and concat optimization issues.
  - Fixed Conv stride 3 corner case hang issue.
- Performance improvement:
  - Updated Conv 1x1 stride 2x2 optimization.
  - Enhanced Conv 7x7 performance.
  - Improved padding performance.
  - Enhanced convolution MT fusion.
  - Improved the performance for NCHW layout model.
  - Enhanced the performance for eltwise-like op.
  - Enhanced Conv and eltwise AT fusion.
  - Improved the output convolution/transpose-convolution's performance.
  - Enhanced the logging message for EoU.

ONNX Runtime EP

- End-2-End Application support on NPU
  - Enhanced existing support: Provided high-level APIs to enable seamless incorporation of pre/post-processing operations into the model to run on NPU
  - Two examples (resnet50 and yolov8) published to demonstrate the usage of these APIs to run end-to-end models on the NPU
- Bug fixes for ONNXRT EP to support customers' models

Misc

- Contains mitigation for the following CVEs: CVE-2024-21974, CVE-2024-21975, CVE-2024-21976

Version 1.0.1

- Minor fix for Single click installation without given env name.
- Perform improvement in the NPU driver.
- Bug fix in elementwise subtraction in the compiler.
- Runtime stability fixes for minor corner cases.
- Quantizer update to resolve performance drop with default settings.

Version 1.0

Quantizer

- ONNX Quantizer
  - Support for ONNXRuntime 1.16.
  - Support for the Cross-Layer-Equalization (CLE) algorithm in quantization, which can balance the weights of consecutive Conv nodes to make it more quantize-friendly in per-tensor quantization.
  - Support for mixed precision quantization including UINT16/INT16/UINT32/INT32/FLOAT16/BFLOAT16, and support asymmetric quantization for BFLOAT16.
  - Support for the MinMSE method for INT16/UINT16/INT32/UINT32 quantization.
  - Support for quantization using the INT16 scale.
  - Support for unsigned ReLU in symmetric activation configuration.
  - Support for converting Float16 to Float32 during quantization.

- Support for converting NCHW model to NHWC model during quantization.
- Support for two more modes for MinMSE for better accuracy. The "All" mode computes the scales with all batches while the "MostCommon" mode computes the scale for each batch and uses the most common scales.
- Support for the quantization of more operations:
  * PReLU, Sub, Max, DepthToSpace, SpaceToDepth, Slice, InstanceNormalization, and LpNormalization.
  * Non-4D ReduceMean.
  * Leakyrelu with arbitrary alpha.
  * Split by converting it to Slice.
- Support for op fusing of InstanceNormalization and L2Normalization in NPU workflow.
- Support for converting Clip to ReLU when the minimal value is 0.
- Updated shift_bias, shift_read, and shift_write constraints in the NPU workflow and added an option "IPULimitationCheck" to disable it.
- Support for disabling the op fusing of Conv + LeakyReLU/PReLU in the NPU workflow.
- Support for logging for quantization configurations and summary information.
- Support for removing initializer from input to support models converted from old version pytorch where weights are stored as inputs.
- Added a recommended configuration for the IPU_Transformer platform.
- New utilities:
  * Tool for converting the float16 model to the float32 model.
  * Tool for converting the NCHW model to the NHWC model.
  * Tool for quantized models with random input.
- Three examples for quantization models from Timm, Torchvision, and ONNXRuntime modelzoo respectively.
- Bugfixes:
  * Fix a bug that weights are quantized with the "NonOverflow" method when using the "MinMSE" method.

- Pytorch Quantizer
  - Support of some operations quantization in quantizer: inplace div, inplace sub
  - Log and document enhancement to emphasize fast-finetune
  - Timm models quantization script example

  - Bug fix for operators: clamp and prelu
  - QAT Support quantization of operations with multiple outputs
  - QAT EOU enhancements: significantly reduces the need for network modifications
  - QAT ONNX exporting enhancements: support more configurations
  - New QAT examples
- TF2 Quantizer
  - Support for Tensorflow 2.11 and 2.12.
  - Support for the 'tf.linalg.matmul' operator.
  - Updated shift_bias constraints for NPU workflow.
  - Support for dumping models containing operations with multiple outputs.
  - Added an example of a sequential model.
  - Bugfixes:
    * Fix a bug that Hardsigmoid and Hardswish are not mapped to DPU without Batch Normalization.
    * Fix a bug when both align_pool and align_concat are used simultaneously.
    * Fix a bug in the sequential model when a layer has multiple consumers.
- TF1 Quantizer
  - Update shift_bias constraints for NPU workflow.
  - Bugfixes:
    * Fix a bug in fast_finetune when the 'input_node' and 'quant_node' are inconsistent.
    * Fix a bug that AddV2 op identified as BiasAdd.
    * Fix a bug when the data type of the concat op is not float.
    * Fix a bug in split_large_kernel_pool when the stride is not equal to 1.

ONNXRuntime Execution Provider

- Support new OPs, such as PRelu, ReduceSum, LpNormlization, DepthToSpace(DCR).
- Increase the percentage of model operators performed on the NPU.
- Fixed some issues causing model operators allocation to CPU.
- Improved report summary
- Support the encryption of the VOE cache
- End-2-End Application support on NPU

  - Enable running pre/post/custom ops on NPU, utilizing ONNX feature of E2E extensions.
  - Two examples published for yolov8 and resnet50, in which preprocessing custom op is added and runs on NPU.
- Performance: latency improves by up to 18% and power savings by up to 35% by additionally running preprocessing on NPU apart from inference.
- Multiple NPU overlays support
  - VOE configuration that supports both CNN-centric and GEMM-centric NPU overlays.
  - Increases number of ops that run on NPU, especially for models which have both GEMM and CNN ops.
  - Examples published for use with some of the vision transformer models.

NPU and Compiler

- New operators support
  - Global average pooling with large spatial dimensions
  - Single Activation (no fusion with conv2d, e.g. relu/single alpha PRelu)
- Operator support enhancement
  - Enlarge the width dimension support range for depthwise-conv2d
  - Support more generic broadcast for element-wise like operator
  - Support output channel not aligned with 4B GStiling
  - Support Mul and LeakyRelu fusion
  - Concatenation's redundant input elimination
  - Channel Augmentation for conv2d (3x3, stride=2)
- Performance optimization
  - PDI partition refine to reduce the overhead for PDI swap
  - Enabled cost model for some specific models
- Fixed asynchronous error in multiple thread scenario
- Fixed known issue on tanh and transpose-conv2d hang issue

Known Issues

- Support for multiple applications is limited to up to eight
- Windows Studio Effects should be disabled when using the Latency profile. To disable Windows Studio Effects, open **Settings > Bluetooth & devices > Camera**, select your primary camera, and then disable all camera effects.

Version 0.9

Quantizer

- Pytorch Quantizer
  - Dict input/output support for model forward function
  - Keywords argument support for model forward function
  - Matmul subroutine quantization support
  - Support of some operations in quantizer: softmax, div, exp, clamp
  - Support quantization of some non-standard conv2d.
- ONNX Quantizer
  - Add support for Float16 and BFloat16 quantization.
  - Add C++ kernels for customized QuantizeLinear and DequantizeLinaer operations.
  - Support saving quantizer version info to the quantized models' producer field.
  - Support conversion of ReduceMean to AvgPool in NPU workflow.
  - Support conversion of BatchNorm to Conv in NPU workflow.
  - Support optimization of large kernel GlobalAvgPool and AvgPool operations in NPU workflow.
  - Supports hardware constraints check and adjustment of Gemm, Add, and Mul operations in NPU workflow.
  - Supports quantization for LayerNormalization, HardSigmoid, Erf, Div, and Tanh for NPU.

ONNXRuntime Execution Provider

- Support new OPs, such as Conv1d, LayerNorm, Clip, Abs, Unsqueeze, ConvTranspose.
- Support pad and depad based on NPU subgraph's inputs and outputs.
- Support for U8S8 models quantized by ONNX quantizer.
- Improve report summary tools.

NPU and Compiler

- Supported exp/tanh/channel-shuffle/pixel-unshuffle/space2depth
- Performance uplift of xint8 output softmax
- Improve the partition messages for CPU/DPU
- Improve the validation check for some operators
- Accelerate the speed of compiling large models
- Fix the elew/pool/dwc/reshape mismatch issue and fix the stride_slice hang issue
- Fix str_w != str_h issue in Conv

LLM

- Smoothquant for OPT1.3b, 2.7b, 6.7b, 13b models.
- Huggingface Optimum ORT Quantizer for ONNX and Pytorch dynamic quantizer for Pytorch
- Enabled Flash attention v2 for larger prompts as a custom torch.nn.Module
- Enabled all CPU ops in bfloat16 or float32 with Pytorch
- int32 accumulator in AIE (previously int16)
- DynamicQuantLinear op support in ONNX
- Support different compute primitives for prefill/prompt and token phases
- Zero copy of weights shared between different op primitives
- Model saving after quantization and loading at runtime for both Pytorch and ONNX
- Enabled profiling prefill/prompt and token time using local copy of OPT Model with additional timer instrumentation
- Added demo mode script with greedy, stochastic and contrastive search options

ASR

- Support Whipser-tiny
- All GEMMs offloaded to AIE
- Improved compile time
- Improved WER

Known issues

- Flow control OPs including “Loop”, “If”, “Reduce” not supported by VOE
- Resizing OP in ONNX opset 10 or lower is not supported by VOE
- Tensorflow 2.x quantizer supports models within tf.keras.model only
- Running quantizer docker in WSL on Ryzen AI laptops may encounter OOM (Out-of-memory) issue
- Running multiple concurrent models using temporal sharing on the 5x4 binary is not supported
- Only batch sizes of 1 are supported
- Only models with the pretrained weights setting = TRUE should be imported
- Launching multiple processes on 4 1x4 binaries can cause hangs, especially when models have many sub-graphs

Version 0.8

Quantizer

- Pytorch Quantizer
  - Pytorch 1.13 and 2.0 support
  - Mixed precision quantization support, supporting float32/float16/bfloat16/intx mixed quantization
  - Support of bit-wise accuracy cross check between quantizer and ONNX-runtime
  - Split and chunk operators were automatically converted to slicing
  - Add support for BFP data type quantization
  - Support of some operations in quantizer: where, less, less_equal, greater, greater_equal, not, and, or, eq, maximum, minimum, sqrt, Elu, Reduction_min, argmin
  - QAT supports training on multiple GPUs
  - QAT supports operations with multiple inputs or outputs
- ONNX Quantizer
  - Provided Python wheel file for installation

  - Support OnnxRuntime 1.15
  - Supports setting input shapes of random data reader
  - Supports random data reader in the dump model function
  - Supports saving the S8S8 model in U8S8 format for NPU
  - Supports simulation of Sigmoid, Swish, Softmax, AvgPool, GlobalAvgPool, ReduceMean and LeakyRelu for NPU
  - Supports node fusions for NPU

ONNXRuntime Execution Provider

- Supports for U8S8 quantized ONNX models
- Improve the function of falling back to CPU EP
- Improve AIE plugin framework
  - Supports LLM Demo
  - Supports Gemm ASR
  - Supports E2E AIE acceleration for Pre/Post ops
  - Improve the easy-of-use for partition and deployment
- Supports models containing subgraphs
- Supports report summary about OP assignment
- Supports report summary about DPU subgraphs falling back to CPU
- Improve log printing and troubleshooting tools.
- Upstreamed to ONNX Runtime Github repo for any data type support and bug fix

NPU and Compiler

- Extended the support range of some operators
  - Larger input size: conv2d, dwc
  - Padding mode: pad
  - Broadcast: add
  - Variant dimension (non-NHWC shape): reshape, transpose, add
- Support new operators, e.g. reducemax(min/sum/avg), argmax(min)
- Enhanced multi-level fusion
- Performance enhancement for some operators

- Add quantization information validation
- Improvement in device partition
  - User friendly message
  - Target-dependency check

Demos

- New Demos link: https://account.amd.com/en/forms/downloads/ryzen-ai-software-platform-xef.html?filename=transformers_2308.zip
  - LLM demo with OPT-1.3B/2.7B/6.7B
  - Automatic speech recognition demo with Whisper-tiny

Known issues

- Flow control OPs including “Loop”, “If”, “Reduce” not supported by VOE
- Resize OP in ONNX opset 10 or lower not supported by VOE
- Tensorflow 2.x quantizer supports models within tf.keras.model only
- Running quantizer docker in WSL on Ryzen AI laptops may encounter OOM (Out-of-memory) issue
- Run multiple concurrent models by temporal sharing on the Performance optimized overlay (5x4.xclbin) is not supported
- Support batch size 1 only for NPU

Version 0.7

Quantizer

- Docker Containers
  - Provided CPU dockers for Pytorch, Tensorflow 1.x, and Tensorflow 2.x quantizer
  - Provided GPU Docker files to build GPU dockers
- Pytorch Quantizer
  - Supports multiple output conversion to slicing
  - Enhanced transpose OP optimization

  - Inspector support new IP targets for NPU
- ONNX Quantizer
  - Provided Python wheel file for installation
  - Supports quantizing ONNX models for NPU as a plugin for the ONNX Runtime native quantizer
  - Supports power-of-two quantization with both QDQ and QOP format
  - Supports Non-overflow and Min-MSE quantization methods
  - Supports various quantization configurations in power-of-two quantization in both QDQ and QOP format.
    * Supports signed and unsigned configurations.
    * Supports symmetry and asymmetry configurations.
    * Supports per-tensor and per-channel configurations.
  - Supports bias quantization using int8 datatype for NPU.
  - Supports quantization parameters (scale) refinement for NPU.
  - Supports excluding certain operations from quantization for NPU.
  - Supports ONNX models larger than 2GB.
  - Supports using CUDAExecutionProvider for calibration in quantization
  - Open source and upstreamed to Microsoft Olive Github repo
- TensorFlow 2.x Quantizer
  - Added support for exporting the quantized model ONNX format.
  - Added support for the keras.layers.Activation('leaky_relu')
- TensorFlow 1.x Quantizer
  - Added support for folding Reshape and ResizeNearestNeighbor operators.
  - Added support for splitting Avgpool and Maxpool with large kernel sizes into smaller kernel sizes.
  - Added support for quantizing Sum, StridedSlice, and Maximum operators.
  - Added support for setting the input shape of the model, which is useful in deploying models with undefined input shapes.
  - Add support for setting the opset version in exporting ONNX format

ONNX Runtime Execution Provider

- Vitis ONNX Runtime Execution Provider (VOE)
  - Supports ONNX Opset version 18, ONNX Runtime 1.16.0, and ONNX version 1.13
  - Supports both C++ and Python APIs(Python version 3)
  - Supports deploy model with other EPs
  - Supports falling back to CPU EP
  - Open source and upstreamed to ONNX Runtime Github repo
  - Compiler
    - Multiple Level op fusion
    - Supports the same muti-output operator like chunk split
    - Supports split big pooling to small pooling
    - Supports 2-channel writeback feature for Hard-Sigmoid and Depthwise-Convolution
    - Supports 1-channel GStiling
    - Explicit pad-fix in CPU subgraph for 4-byte alignment
    - Tuning the performance for multiple models

NPU

- Two configurations
  - Power Optimized Overlay
    - Suitable for smaller AI models (1x4.xclbin)
    - Supports spatial sharing, up to 4 concurrent AI workloads
  - Performance Optimized Overlay (5x4.xclbin)
    - Suitable for larger AI models

Known issues

- Flow control OPs including “Loop”, “If”, “Reduce” are not supported by VOE
- Resize OP in ONNX opset 10 or lower not supported by VOE
- Tensorflow 2.x quantizer supports models within tf.keras.model only
- Running quantizer docker in WSL on Ryzen AI laptops may encounter OOM (Out-of-memory) issue

- Run multiple concurrent models by temporal sharing on the Performance optimized overlay (5x4.xclbin) is not supported

2.2.2 Installation Instructions

Prerequisites

The Ryzen AI Software supports AMD processors with a Neural Processing Unit (NPU). Refer to the release notes for the full list of *supported configurations*.

The following dependencies must be installed on the system before installing the Ryzen AI Software:

Dependencies	Version Requirement
Windows 11	build >= 22621.3527
Visual Studio	2022
cmake	version >= 3.26
Python distribution (Miniforge preferred)	Latest version

IMPORTANT:

- Visual Studio 2022 Community: ensure that *Desktop Development with C++* is installed
- Miniforge: ensure that the following path is set in the System PATH variable: `path\to\miniforge3\condabin` or `path\to\miniforge3\Scripts\` or `path\to\miniforge3\` (The System PATH variable should be set in the *System Variables* section of the *Environment Variables* window).

Install NPU Drivers

- Download the NPU driver installation package `NPU Driver`
- Install the NPU drivers by following these steps:
  - Extract the downloaded ZIP file.
  - Open a terminal in administrator mode and execute the `.\npu_sw_installer.exe` file.

- Ensure that NPU MCDM driver (Version:32.0.203.280, Date:5/16/2025) is correctly installed by opening Task Manager -> Performance -> NPU0.

Install Ryzen AI Software

- Download the RyzenAI Software installer `ryzen-ai-1.5.0.msi`.
- Launch the MSI installer and follow the instructions on the installation wizard:
  - Accept the terms of the Licence agreement
  - Provide the destination folder for Ryzen AI installation (default: `C:\Program Files\RyzenAI\1.5.0`)
  - Specify the name for the conda environment (default: `ryzen-ai-1.5.0`)

The Ryzen AI Software packages are now installed in the conda environment created by the installer.

> **Note**
>
> The latest updates with LLM performance improvements are available in the RAI 1.5.1 release. Download it from the link `ryzen-ai-1.5.1.msi`.

Test the Installation

The Ryzen AI Software installation folder contains test to verify that the software is correctly installed. This installation test can be found in the `quicktest` subfolder.

- Open a Conda command prompt (search for "Miniforge Prompt" in the Windows start menu)
- Activate the Conda environment created by the Ryzen AI installer:

```
conda activate <env_name>
```

- Run the test:

```
cd %RYZEN_AI_INSTALLATION_PATH%/quicktest
python quicktest.py
```

- The quicktest.py script sets up the environment and runs a simple CNN model. On a successful run, you will see an output similar to the one shown below. This indicates that the model is running on the NPU and that the installation of the Ryzen AI Software was successful:

```
[Vitis AI EP] No. of Operators :   NPU   398 VITIS_EP_CPU     2
[Vitis AI EP] No. of Subgraphs :   NPU     1 Actually running on NPU     1
Test Passed
```

Note

The full path to the Ryzen AI Software installation folder is stored in the `RYZEN_AI_INSTALLATION_PATH` environment variable.

2.2.3 Examples, Demos, Tutorials

This page introduces various demos, examples, and tutorials currently available with the Ryzen™ AI Software.

Getting Started Tutorials

NPU

- Getting Started Tutorial for INT8 models - Uses a custom ResNet model to demonstrate:
  - Pretrained model conversion to ONNX
  - Model Quantization using AMD Quark quantizer
  - Deployment using ONNX Runtime C++ and Python code
- Getting Started Tutorial for BF16 models - Uses a custom ResNet model to demonstrate:
  - Preparation and compilation of BF16 models
  - Deployment using Python
  - Deployment using C++
- Hello World Jupyter Notebook Tutorial
- Additional BF16 model examples:
  - Image Classification
  - Finetuned DistilBERT for Text Classification
  - Text Embedding Model Alibaba-NLP/gte-large-en-v1.5

iGPU

- ResNet50 on iGPU

Other examples, demos, and tutorials

- Refer to RyzenAI-SW repo

2.2.4 Model Quantization

Model quantization is the process of mapping high-precision weights/activations to a lower precision format, such as BF16/INT8, while maintaining model accuracy. This technique enhances the computational and memory efficiency of the model for deployment on NPU devices. It can be applied post-training, allowing existing models to be optimized without the need for retraining.

The Ryzen AI compiler supports input models quantized to either INT8 or BF16 format:

- CNN models: INT8 or BF16
- Transformer models: BF16

Quantization introduces several challenges, primarily revolving around the potential drop in model accuracy. Choosing the right quantization parameters—such as data type, bit-width, scaling factors, and the decision between per-channel or per-tensor quantization—adds layers of complexity to the design process. The list of operations supported for different quantization types can be found in Supported Operations.

AMD Quark

AMD Quark is a comprehensive cross-platform deep learning toolkit designed to simplify and enhance the quantization of deep learning models. Supporting both PyTorch and ONNX models, Quark empowers developers to optimize their models for deployment on a wide range of hardware backends, achieving significant performance gains without compromising accuracy.

For more challenging model quantization needs **AMD Quark** supports advanced quantization technique like **Fast Finetuning** that helps recover the lost accuracy of the quantized model.

Documentation

The complete documentation for AMD Quark for Ryzen AI can be found here: https://quark.docs.amd.com/latest/supported_accelerators/ryzenai/index.html

INT8 Examples

AMD Quark provides default configrations that support INT8 quantization configuration. For example, *XINT8* uses symmetric INT8 activation and weights quantization with power-of-two scales using the MinMSE calibration method. The quantization configuration can be customized using the *QuantizationConfig* class. The following example shows how to set up the quantization configuration for INT8 quantization:

```
quant_config = QuantizationConfig(calibrate_method=PowerOfTwoMethod.
↪MinMSE,
                                  activation_type=QuantType.QUInt8,
                                  weight_type=QuantType.QInt8,
                                  enable_npu_cnn=True,
```

(continues on next page)

(continued from previous page)

```
                                    extra_options={'ActivationSymmetric':␣
↪True})
config = Config(global_quant_config=quant_config)
print("The configuration of the quantization is {}".format(config))
```

The user can use the *get_default_config('XINT8')* function to get the default configuration for INT8 quantization.

For more details

- AMD Quark Tutorial for Ryzen AI Deployment
- Running INT8 model on NPU using Getting Started Tutorial
- Advanced quantization techniques Fast Finetuning and Cross Layer Equalization for INT8 model

BF16 Examples

AMD Quark provides default configrations that support BFLOAT16 (BF16) model quantization. For example, BF16 is a 16-bit floating-point format designed to have same exponent size as FP32, allowing a wide dynamic range, but with reduced precision to save memory and speed up computations. The BFLOAT16 (BF16) model needs to be converted from QDQ nodes to Cast operations to run with VAIML compiler. AMD Quark support this conversion with the configuration option *BF16QDQToCast*.

```
quant_config = get_default_config("BF16")
quant_config.extra_options["BF16QDQToCast"] = True
config = Config(global_quant_config=quant_config)
print("The configuration of the quantization is {}".format(config))
```

For more details

- Image Classification using ResNet50 to run BF16 model on NPU
- Finetuned DistilBERT for Text Classification
- Text Embedding Model Alibaba-NLP/gte-large-en-v1.5
- Advanced quantization techniques Fast Finetuning for BF16 models.

2.2.5 Model Compilation and Deployment

Introduction

The Ryzen AI Software supports models saved in the ONNX format and uses ONNX Runtime as the primary mechanism to load, compile and run models.

NOTE: Models with ONNX opset 17 are recommended. If your model uses a different opset version, consider converting it using the ONNX Version Converter

For a complete list of supported operators, consult this page: Supported Operators.

Loading Models

Models are loaded by creating an ONNX Runtime `InferenceSession` using the Vitis AI Execution Provider (VAI EP):

```
import onnxruntime

session_options = onnxruntime.SessionOptions()
vai_ep_options  = {}                            # Vitis AI EP options go␣
↪here

session = onnxruntime.InferenceSession(
    path_or_bytes = model,                      # Path to the ONNX model
    sess_options = session_options,             # Standard ORT options
    providers = ['VitisAIExecutionProvider'], # Use the Vitis AI␣
↪Execution Provider
    provider_options = [vai_ep_options]         # Pass options to the Vitis␣
↪AI Execution Provider
)
```

The `provider_options` parameter enables the configuration of the Vitis AI Execution Provider (EP). For a comprehensive list of supported provider options, refer to the *Vitis AI EP Options Reference Guide* below.

When a model is first loaded into an ONNX Runtime (ORT) inference session, it is compiled into the format required by the NPU. The resulting compiled output can be saved as an ORT EP context file or stored in the Vitis AI EP cache directory.

If a compiled version of the ONNX model is already available — either as an EP context file or within the Vitis AI EP cache — the model will not be recompiled. Instead, the precompiled version will be loaded automatically. This greatly reduces session creation time and improves overall efficiency. For more details, refer to the section on *Managing Compiled Models*.

Deploying Models

Once the ONNX Runtime inference session is initialized and the model is compiled, the model is deployed using the ONNX Runtime `run()` API:

```
input_data = {}
for input in session.get_inputs():
    input_data[input.name] = ...  # Initialize input tensors

outputs = session.run(None, input_data) # Run the model
```

The ONNX graph is automatically partitioned into multiple subgraphs by the Vitis AI Execution Provider (EP). During deployment, the subgraph(s) containing operators supported by the NPU are executed on the NPU. The remaining subgraph(s) are executed on the CPU. This graph partitioning and deployment technique across CPU and NPU is fully automated by the VAI EP and is totally transparent to the end-user.

Vitis AI EP Options Reference Guide

VitisAI EP Provider Options

The `provider_options` parameter of the ORT `InferenceSession` allows passing options to configure the Vitis AI EP. The following options are supported.

- **`config_file`**

Optional. Configuration file to pass additional compilation options for BF16 models. For more details, refer to the section about *Config File Options*.

Type: String

Default: N/A

- **`xclbin`**

Required for INT8 models. NPU binary file to specify NPU configuration to be used for INT8 models. For more details, refer to the section about *Using INT8 Models*.

Type: String

Default: N/A.

- **`encryption_key`**

Optional. 256-bit key used for encrypting the EP context model. At runtime, you must use the same key to decrypt the model when loading it. For more details, refer to the section about the *EP Context Cache* feature.

Type: String of 64 hexadecimal values representing the 256-bit encryption key.

Default: None, the model is not encrypted.

- **opt_level**

Optional. Applies to INT8 models only. Controls the compiler optimization effort.

Supported Values: 0, 1, 2, 3, 65536 (maximum effort, experimental)

Default: 0

- **log_level**

Optional. Controls what level of messages are reported by the VitisAI EP.

Supported Values: "info", "warning", "warning", "error", "fatal"

Default: "info"

- **cache_dir**

Optional. The path and name of the VitisAI cache directory. For INT8 models, for this option to take affect, the *enable_cache_file_io_in_mem* must be set to 0. For more details, refer to the section about *VitisAI cache*.

Type: String

Default: C:\temp\%USERNAME%\vaip\.cache

- **cache_key**

Optional. The subfolder in the VitisAI cache directory where the compiled model is stored. For INT8 models, for this option to take affect, the *enable_cache_file_io_in_mem* must be set to 0. For more details, refer to the section about *VitisAI cache*.

Type: String

Default: MD5 hash of the input model.

- **enable_cache_file_io_in_mem**

Optional. Applies to INT8 models only. By default, the VitisAI EP keeps the compiled model in memory. To enable saving the compiled model to disk in the *cache_dir* folder, *enable_cache_file_io_in_mem* must be set to 0.

Supported Values: 0, 1

Default: 1

- **ai_analyzer_visualization**

Optional. Enables generation of compile-time analysis data.

Type: Boolean

Default: False

- **ai_analyzer_profiling**

Optional. Enables generation of inference-time analysis data.

Type: Boolean

Default: False

Config File Options

When compiling BF16 models, a JSON configuration file can be provided to the VitisAI EP using the *config_file* provider option. This configuration file is used to specify additional options to the compiler.

The default the configuration file for compiling BF16 models contains the following:

```
{
 "passes": [
     {
         "name": "init",
         "plugin": "vaip-pass_init"
     },
     {
         "name": "vaiml_partition",
         "plugin": "vaip-pass_vaiml_partition",
         "vaiml_config": {
             "optimize_level": 1,
             "preferred_data_storage": "auto"
         }
     }
 ]
}
```

The `vaiml_config` section of the configuration file contains the user options. The supported user options are described below.

- **optimize_level**

Controls the compiler optimization level.

Supported values: 1 (default), 2, 3

- **preferred_data_storage**

Controls whether intermediate data is stored in vectorized or unvectorized format. Models dominated by convolutions (e.g., CNNs) perform better with vectorized data. Models dominated by GEMMs (e.g., Transformers) perform better with unvectorized data. By default ("auto") the compiler tries to select the best layout.

Supported values: "auto" (default), "vectorized", "unvectorized"

Using BF16 models

When compiling BF16 models, a configuration file must be provided to the VitisAI EP. This file is specified using the *config_file* provider option. For more details, refer to *Config File Options* section.

Sample Python Code

Python example loading a configuration file called vai_ep_config.json:

```
import onnxruntime

vai_ep_options = {
    'config_file': 'vai_ep_config.json'
}

session = onnxruntime.InferenceSession(
    "resnet50_bf16.onnx",
    providers=['VitisAIExecutionProvider'],
    provider_options=[vai_ep_options]
)
```

Sample C++ Code

C++ example loading a configuration file called vai_ep_config.json:

```
#include <onnxruntime_cxx_api.h>

auto onnx_model = "resnet50_bf16.onnx"
Ort::Env env(ORT_LOGGING_LEVEL_WARNING, "resnet50_bf16");
auto session_options = Ort::SessionOptions();
auto vai_ep_options = std::unorderd_map<std::string,std::string>({});
vai_ep_options["config_file"] = "vai_ep_config.json";
session_options.AppendExecutionProvider_VitisAI(vai_ep_options);
auto session = Ort::Session(
    env,
    std::basic_string<ORTCHAR_T>(onnx_model.begin(), onnx_model.end()).c_
↪str(),
    session_options);
```

Using INT8 models

When compiling INT8 models, the NPU configuration must be specified through the *xclbin* provider option. This option is not required for BF16 models.

Setting the NPU configuration involves specifying one of `.xclbin` binary files located in the Ryzen AI Software installation tree.

It is recommended to copy the required xclbin files from the Ryzen AI installation tree into the project folder as the xclbin files used to compile the model must be included in the final version of the application.

Depending on the target processor type, the following `.xclbin` files should be used:

For STX/KRK APUs:

- `%RYZEN_AI_INSTALLATION_PATH%\voe-4.0-win_amd64\xclbins\strix\AMD_AIE2P_4x4_Overlay.xclbin`

For PHX/HPT APUs:

- `%RYZEN_AI_INSTALLATION_PATH%\voe-4.0-win_amd64\xclbins\phoenix\4x4.xclbin`

NOTE: Starting in Ryzen AI 1.5, the legacy "1x4" and "Nx4" xclbin files are no longer supported and should not be used.

Sample Python Code

Python example selecting the `AMD_AIE2P_4x4_Overlay.xclbin` NPU configuration for STX/KRK located in the Ryzen AI installation folder:

```
import os
import onnxruntime

vai_ep_options = {
    'xclbin': os.path.join(os.environ['RYZEN_AI_INSTALLATION_PATH'], 'voe-
↪4.0-win_amd64', 'xclbins', 'strix', 'AMD_AIE2P_4x4_Overlay.xclbin')
}

session = onnxruntime.InferenceSession(
    "resnet50_int8.onnx",
    providers=['VitisAIExecutionProvider'],
    provider_options=[vai_ep_options]
)
```

Sample C++ Code

C++ example selecting the `AMD_AIE2P_4x4_Overlay.xclbin` NPU configuration for STX/KRK located in a custom folder:

```
#include <onnxruntime_cxx_api.h>

auto onnx_model = "resnet50_int8.onnx"
Ort::Env env(ORT_LOGGING_LEVEL_WARNING, "resnet50_int8");
auto session_options = Ort::SessionOptions();
auto vai_ep_options = std::unorderd_map<std::string,std::string>({});
vai_ep_options["xclbin"] = "/path/to/xclbins/strix/AMD_AIE2P_4x4_Overlay.
↪xclbin";
session_options.AppendExecutionProvider_VitisAI(vai_ep_options);
auto session = Ort::Session(
    env,
    std::basic_string<ORTCHAR_T>(onnx_model.begin(), onnx_model.end()).c_
↪str(),
    session_options);
```

Managing Compiled Models

To avoid the overhead of recompiling models, it is very advantageous to save the compiled models and use these pre-compiled versions in the final application. Pre-compiled models can be loaded instantaneously and immediately executed on the NPU. This greatly improves the session creation time and overall end-user experience.

The RyzenAI Software supports two mechanisms for saving and reloading compiled models:

- VitisAI EP Cache
- ONNX Runtime EP Context Cache

TIP: The VitisAI EP Cache mechanism is most convenient to quickly iterate during the development cycle. The OnnxRuntime EP Context Cache mechanism is recommended for the final version of the application.

VitisAI EP Cache

The VitisAI EP includes a built-in caching mechanism. When a model is compiled for the first time, it is automatically saved in the VitisAI EP cache directory. Any subsequent creation of an ONNX Runtime session using the same model will load the precompiled model from the cache directory, thereby reducing session creation time.

The VitisAI EP Cache mechanism can be used to quickly iterate during the development cycle, but it is not recommended for the final version of the application.

Cache directories generated by the Vitis AI Execution Provider should not be reused across different versions of the Vitis AI EP or across different version of the NPU drivers.

If using the VitisAI EP Cache the application should check the version of the Vitis AI EP and of the NPU drivers. If the application detects a version change, it should delete the cache, or create a new cache directory with a different name.

The location of the VitisAI EP cache is specified with the *cache_dir* and *cache_key* provider options. For INT8 models, the *enable_cache_file_io_in_mem* must be set to 0 otherwise the output of the compiler is kept in memory and is not saved to disk.

Python example:

```
import onnxruntime
from pathlib import Path

vai_ep_options = {
    'cache_dir': str(Path(__file__).parent.resolve()),
    'cache_key': 'compiled_resnet50_int8',
    'enable_cache_file_io_in_mem': 0
}

session = onnxruntime.InferenceSession(
    "resnet50_int8.onnx",
    providers=['VitisAIExecutionProvider'],
    provider_options=[vai_ep_options]
)
```

In the example above, the cache directory is set to the absolute path of the folder containing the script being executed. Once the session is created, the compiled model is saved inside a subdirectory named `compiled_resnet50_int8` within the specified cache folder.

ONNX Runtime EP Context Cache

The Vitis AI EP supports the ONNX Runtime EP context cache feature. This features allows dumping and reloading a snapshot of the EP context before deployment.

The user can enable dumping of the EP context by setting the `ep.context_enable` session option to 1.

The following options can be used for additional control:

- `ep.context_file_path` – Specifies the output path for the dumped context model.
- `ep.context_embed_mode` – Embeds the EP context into the ONNX model when set to 1.

For further details, refer to the official ONNX Runtime documentation: https://onnxruntime.ai/docs/execution-providers/EP-Context-Design.html

EP Context Encryption

By default, the generated context model is unencrypted and can be used directly during inference. If needed, the context model can be encrypted using one of the methods described below.

User-managed encryption

After the context model is generated, the developer can encrypt the generated file using a method of choice. At runtime, the encrypted file can be loaded by the application, decrypted in memory and passed as a serialized string to the inference session.

This method gives complete control to the developer over the encryption process.

EP-managed encryption

The VitisAI EP can optionally encrypt the EP context model using AES256. This is enabled by passing an encryption key using the *encryption_key* VAI EP provider options. The key is a 256-bit value represented as a 64-digit string. At runtime, the same encryption key must be provided to decrypt and load the context model.

With this method, encryption and decryption is seamlessly managed by the VitisAI EP.

Python example:

```
import onnxruntime

vai_ep_options = {
    'xclbin': r'/path/to/xclbins/strix/AMD_AIE2P_4x4_Overlay.xclbin'),
    'encryptionKey':
↪'89703f950ed9f738d956f6769d7e45a385d3c988ca753838b5afbc569ebf35b2'
}

# Compilation session
session_options = ort.SessionOptions()
session_options.add_session_config_entry('ep.context_enable', '1')
session_options.add_session_config_entry('ep.context_file_path', 'context_
↪model.onnx')
session_options.add_session_config_entry('ep.context_embed_mode', '1')
session = ort.InferenceSession(
    path_or_bytes='resnet50_int8.onnx',  # Load the ONNX model
    sess_options=session_options,
    providers=['VitisAIExecutionProvider'],
    provider_options=[vai_ep_options]
```

(continues on next page)

(continued from previous page)

```
)

# Inference session
session_options = ort.SessionOptions()
session = ort.InferenceSession(
    path_or_bytes='context_model.onnx', # Load the EP context model
    sess_options=session_options,
    providers=['VitisAIExecutionProvider'],
    provider_options=[vai_ep_options]
)
```

C++ example:

```
Ort::Env env(ORT_LOGGING_LEVEL_WARNING, "ort");

// VAI EP Provider options
auto vai_ep_options = std::unorderd_map<std::string,std::string>({});
vai_ep_options["xclbin"] = "/path/to/xclbins/strix/AMD_AIE2P_4x4_Overlay.
↪xclbin";
vai_ep_options["encryption_key"] =
↪"89703f950ed9f738d956f6769d7e45a385d3c988ca753838b5afbc569ebf35b2";

// Session options
auto session_options = Ort::SessionOptions();
session_options.AppendExecutionProvider_VitisAI(vai_ep_options);

// Inference session
auto onnx_model = "context_model.onnx"; // The EP context model
auto session = Ort::Session(
    env,
    std::basic_string<ORTCHAR_T>(onnx_model.begin(), onnx_model.end()).c_
↪str(),
    session_options);
```

NOTE: It is possible to precompile the EP context model using Python and to deploy it using a C++ program.

Operator Assignment Report

The compiler can optionally generate a report on operator assignments across CPU and NPU. To generate this report:

- The *enable_cache_file_io_in_mem* provider option must be set to 0
- The XLNX_ONNX_EP_REPORT_FILE environment variable must be used to specify the name of the generated report. For instance:

```
set XLNX_ONNX_EP_REPORT_FILE=vitisai_ep_report.json
```

When these conditions are satisfied, the report file is automatically generated in the cache directory. This report includes information such as the total number of nodes, the list of operator types in the model, and which nodes and operators run on the NPU or on the CPU. Additionally, the report includes node statistics, such as input to a node, the applied operation, and output from the node.

When these conditions are satisified, the report file is automatically generated in the cache directory. This report includes information such as the total number of nodes, the list of operator types in the model, and which nodes and operators runs on the NPU or on the CPU. Additionally, the report includes node statistics, such as input to a node, the applied operation, and output from the node.

```
{
 "deviceStat": [
 {
  "name": "all",
  "nodeNum": 400,
  "supportedOpType": [
   "::Add",
   "::Conv",
   ...
  ]
 },
 {
  "name": "CPU",
  "nodeNum": 2,
  "supportedOpType": [
   "::DequantizeLinear",
   "::QuantizeLinear"
  ]
 },
 {
  "name": "NPU",
  "nodeNum": 398,
  "supportedOpType": [
   "::Add",
```

(continues on next page)

(continued from previous page)

```
  "::Conv",
  ...
 ]
 ...
```

To disable generation of the report, unset the XLNX_ONNX_EP_REPORT_FILE environment variable:

```
set XLNX_ONNX_EP_REPORT_FILE=
```

2.2.6 Application Development

This page captures requirements and recommendations for developers looking to create, package and distribute applications targeting NPU-enabled AMD processors.

VitisAI EP / NPU Driver Compatibility

The VitisAI EP requires a compatible version of the NPU drivers. For each version of the VitisAI EP, compatible drivers are bounded by a minimum version and a maximum release date. NPU drivers are backward compatible with VitisAI EP released up to three years. The maximum driver release date is therefore set to three years after the release date of the corresponding VitisAI EP.

The following table summarizes the driver requirements for the different versions of the VitisAI EP.

VitisAI EP version	Minimum NPU Driver version	Maximum NPU Driver release date
1.5	32.0.203.280	July 1st, 2028
1.4.1	32.0.203.259	May 13th, 2028
1.4	32.0.203.257	March 25th, 2028
1.3.1	32.0.203.242	January 17th, 2028
1.3	32.0.203.237	November 26th, 2027
1.2	32.0.201.204	July 30th, 2027

The application must verify that NPU drivers compatible with the version of the Vitis AI EP in use are installed.

APU Types

The Ryzen AI Software supports various types of NPU-enabled APUs, referred to as PHX, HPT, STX, and KRK. To programmatically determine the type of the local APU, you can enumerate the PCI devices and look for an instance with a matching Hardware ID.

Vendor	Device	Revision	APU Type
0x1022	0x1502	0x00	PHX or HPT
0x1022	0x17F0	0x00	STX
0x1022	0x17F0	0x10	STX
0x1022	0x17F0	0x11	STX
0x1022	0x17F0	0x20	KRK

The application must verify that it is running on an AMD processor with an NPU, and that the NPU type is supported by the version of the Vitis AI EP in use.

Application Development Requirements

ONNX-RT Session

The application should only use the Vitis AI Execution Provider if the following conditions are met:

- The application is running on an AMD processor with an NPU type supported by the version of the Vitis AI EP in use. See *list*.
- NPU drivers compatible with the version of the Vitis AI EP being used are installed. See *compatibility table* .

NOTE: Sample C++ code that implements the compatibility checks to be performed before using the Vitis AI EP is available here

VitisAI EP Provider Options

For INT8 models, the application should detect the type of APU present (PHX, HPT, STX, or KRK) and set the `xclbin` provider option accordingly. Refer to the section on *using INT8 models* for more details.

For BF16 models, the application should set the `config_file` provider option to the same file that was used to precompile the BF16 model. Refer to the section on *using BF16 models* for more details.

Pre-Compiled Models

To avoid the overhead of recompiling models, it is highly recommended to save the compiled models and use these precompiled versions in the final application. Precompiled models can be loaded instantly and executed immediately on the NPU, significantly improving session creation time and overall end-user experience.

AMD recommends using the ONNXRuntime *EP Context Cache* feature for saving and reloading compiled models.

BF16 models

The deployment version of the VitisAI Execution Provider (EP) does not support the on-the-fly compilation of BF16 models. Applications utilizing BF16 models must include pre-compiled versions of these models. The VitisAI EP can then load the pre-compiled models and deploy them efficiently on the NPU.

INT8 models

Including pre-compiled versions of INT8 models is recommended but not mandatory.

Application Packaging Requirements

A C++ application built on the Ryzen AI ONNX Runtime must include the following components in its distribution package:

For INT8 models

- DLLs:
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_providers_shared.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_providers_vitisai.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_vitisai_ep.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\dyn_dispatch_core.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\xaiengine.dll
- NPU Binary files (.xclbin) from the `%RYZEN_AI_INSTALLATION_PATH%\voe-4.0-win_amd64\xclbins` folder
- Recommended but not mandatory: pre-compiled models in the form of *Onnx Runtime EP context models*

For BF16 models

- DLLs:
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_providers_shared.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_providers_vitisai.dll

  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_vitisai_ep.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\dyn_dispatch_core.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\xaiengine.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\flexmlrt.dll
- Pre-compiled models in the form of *Vitis AI EP cache folders*

For Hybrid LLMs

- DLLs:
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime-genai.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\ryzen_mm.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\dyn_dispatch_core.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\xaiengine.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnx_custom_ops.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\libutf8_validity.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\abseil_dll.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\DirectML.dll

For NPU-only LLMs

- DLLs:
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime-genai.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\ryzen_mm.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\dyn_dispatch_core.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\xaiengine.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_providers_shared.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_providers_vitisai.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_vitis_ai_custom_ops.dll
  - %RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_vitisai_ep.dll
- VAIP LLM configuration file: %RYZEN_AI_INSTALLATION_PATH%\deployment\vaip_llm.json

2.2.7 Overview

LLM Deployment on Ryzen AI

Large Language Models (LLMs) can be deployed on Ryzen AI PCs with NPU and GPU acceleration. NPU-only and Hybrid execution modes, which utilize both the NPU and integrated GPU (iGPU), are supported via ONNXRuntime GenAI (OGA). GPU-only acceleration is enabled through llama.cpp. See the *LLM Execution Mode Comparison* below for detailed information.

Execution Modes

Table 1: LLM Execution Mode Comparison

Mode	Framework(s)	Compute Allocation	Primary Use Case
NPU-Only	OnnxRuntime GenAI (OGA)	Neural Processing Unit (NPU) exclusive	Maximum NPU utilization while preserving iGPU for parallel workloads
Hybrid	OnnxRuntime GenAI (OGA)	Dynamic NPU + iGPU partitioning	Interactive inference with optimal prefill/decode performance
GPU	llama.cpp	Dedicated GPU execution	High-throughput inference on discrete/integrated GPU
CPU	OGA or llama.cpp	Traditional CPU-based inference	Baseline compatibility across all processor generations

Hardware Requirements

Table 2: Supported Processor Configurations

Processor Series	NPU-Only	Hybrid	GPU/CPU
Ryzen AI 300 (STX/KRK)	✓	✓	✓
Ryzen AI 7000/8000			✓

Development Interfaces

The Ryzen AI LLM software stack is available through three development interfaces, each suited for specific use cases as outlined in the sections below. All three interfaces are built on top of native OnnxRuntime GenAI (OGA) libraries or llama.cpp libraries, as shown in the *Ryzen AI Software Stack* diagram below.

The high-level Python APIs, as well as the Server Interface, also leverage the Lemonade SDK, which is multi-vendor open-source software that provides everything necessary for quickly getting started with LLMs on OGA or llama.cpp.

A key benefit of Lemonade is that software developed against their interfaces is portable to many other execution backends.

Table 3: Ryzen AI Software Stack

Your Python Application	Your LLM Stack	Your Native Application
*Lemonade Python API**	*Lemonade Server Interface**	OGA C++ Headers **OR** llama.cpp C++ Headers
Custom AMD OnnxRuntime GenAI (OGA) **OR** llama.cpp*		
AMD Ryzen AI Driver and Hardware		

* indicates open-source software (OSS).

Server Interface (REST API)

The Server Interface provides a convenient means to integrate with applications that:

- Already support an LLM server interface, such as the Ollama server or OpenAI API.
- Are written in any language (C++, C#, Javascript, etc.) that supports REST APIs.
- Benefits from process isolation for the LLM backend.

Lemonade Server is available in two ways:

- **Standalone Windows GUI installer**: Quick setup with a desktop shortcut for immediate use. (Recommended for end users, see *Server Interface (REST API)*)
- **Full Lemonade SDK**: Complete development toolkit with server interface included. (Recommended for developers, see *High-Level Python SDK* for Python SDK)

For example applications that have been tested with Lemonade Server, see the Lemonade Server Examples.

High-Level Python SDK

The high-level Python SDK, Lemonade, allows you to get started using PyPI installation in approximately 5 minutes.

This SDK allows you to:

- Experiment with models in hybrid or NPU-only execution mode on Ryzen AI hardware.
- Validate inference speed and task performance.

- Integrate with Python apps using a high-level API.

To get started in Python, follow these instructions: *High-Level Python SDK*.

OGA APIs for C++ Libraries and Python

Native C++ libraries for OGA are available to give full customizability for deployment into native applications. The Python bindings for OGA also provide a customizable interface for Python development.

To get started with the OGA APIs, follow these instructions: *OnnxRuntime GenAI (OGA) Flow*.

Supported LLMs

The comprehensive set of pre-optimized models for hybrid execution are available in the AMD hybrid collection on Hugging Face and the NPU-only examples are available in the AMD NPU collection on Hugging Face. It is also possible to run fine-tuned versions of the models listed (for example, fine-tuned versions of Llama2 or Llama3). For instructions on how to prepare a fine-tuned OGA model, refer to *Preparing OGA Models*.

End to End OGA Validation

A Jupyter Notebook example is provided to demonstrate end-to-end validation of OGA hybrid and NPU-only execution. This notebook includes:

- Installation
- Command Syntax
- Benchmarking
- Subjective Evaluation
- Objective Evaluation

To run the notebook, visit the Lemonade Tools Tutorial.

2.2.8 Server Interface (REST API)

The Lemonade SDK offers a server interface that allows your application to load an LLM on Ryzen AI hardware in a process, and then communicate with this process using standard `REST` APIs. This allows applications written in any language (C#, JavaScript, Python, C++, etc.) to easily integrate with Ryzen AI LLMs.

Server interfaces are used across the LLM ecosystem because they allow for no-code plug-and-play between the higher level of the application stack (GUIs, agents, RAG, etc.) with the LLM and hardware that have been abstracted by the server. For more information, see the Understanding local LLM Servers Guide.

For example, open source projects such as *Open WebUI* have out-of-box support for connecting to a variety of server interfaces, which in turn allows users to quickly start working with LLMs in a GUI.

Server Setup

Lemonade Server can be installed via the Lemonade Server Installer executable by following these steps:

1. Make sure your system has the recommended Ryzen AI driver installed as described in *Install NPU Drivers*.
2. Download and install `Lemonade_Server_Installer.exe` from the latest Lemonade release.
3. Launch the server by double-clicking the `lemonade_server` shortcut added to your desktop.

For a visual walkthrough of this process, watch our Lemonade Introductory Video:

See the Lemonade Server Documentation for more details.

Server Usage

The Lemonade Server provides the following OpenAI-compatible endpoints:

- POST `/api/v1/chat/completions` - Chat Completions (messages to completions)
- POST `/api/v1/completions` - Text Completions (prompt to completion)
- POST `/api/v1/responses` - Chat Completions (prompt|messages -> event)
- GET `/api/v1/models` - List available models

Please refer to the server specification document for details about the request and response formats for each endpoint.

The OpenAI API documentation also has code examples for integrating streaming completions into an application.

Supported Applications

The Lemonade Server supports a variety of applications that can connect to it using the OpenAI API. Some of the applications that have been tested with Lemonade Server can be found at Lemonade Server Apps.

A short list of applications that have been tested with Lemonade Server includes:

Next Steps

- See Lemonade Server Examples to find applications that have been tested with Lemonade Server.
- Check out the Lemonade Server specification to learn more about supported features.
- Try out your Lemonade Server install with any application that uses the OpenAI chat completions API.

2.2.9 High-Level Python SDK

A Python environment offers flexibility for experimenting with LLMs, profiling them, and integrating them into Python applications. We use the Lemonade SDK to get up and running quickly.

To get started, follow these instructions.

System-level pre-requisites

You only need to do this once per computer:

1. Make sure your system has the recommended Ryzen AI driver installed as described in *Install NPU Drivers*.
2. Download and install Miniconda for Windows or Miniforge for Windows.
3. Launch a terminal and call `conda init`.

Environment Setup

To create and set up an environment, run these commands in your terminal:

```
conda create -n ryzenai-llm python=3.10
conda activate ryzenai-llm
pip install lemonade-sdk[dev,oga-ryzenai] --extra-index-url=https://pypi.
↪amd.com/simple
```

Validation Tools

Now that you have completed installation, you can try prompting an LLM like this (where `PROMPT` is any prompt you like).

Run this command in a terminal that has your environment activated:

```
lemonade -i amd/Llama-3.2-1B-Instruct-awq-g128-int4-asym-fp16-onnx-hybrid␣
↪oga-load --device hybrid --dtype int4 llm-prompt --max-new-tokens 64 -p␣
↪PROMPT
```

For an end-to-end example demonstrating the Validation Tools, visit the Lemonade Tools Tutorial.

Python API

You can also run this code to try out the high-level Lemonade API in a Python script:

```
from lemonade.api import from_pretrained

model, tokenizer = from_pretrained(
    "amd/Llama-3.2-1B-Instruct-awq-g128-int4-asym-fp16-onnx-hybrid",␣
↪recipe="oga-hybrid"
)

input_ids = tokenizer("This is my prompt", return_tensors="pt").input_ids
response = model.generate(input_ids, max_new_tokens=30)

print(tokenizer.decode(response[0]))
```

Next Steps

From here, you can check out the Jupyter Notebook that provides an end-to-end validation of OGA hybrid and NPU-only execution. To run the notebook, visit the Lemonade Tools Tutorial.

2.2.10 OnnxRuntime GenAI (OGA) Flow

Ryzen AI Software supports deploying LLMs on Ryzen AI PCs using the native ONNX Runtime Generate (OGA) C++ or Python API. The OGA API is the lowest-level API available for building LLM applications on a Ryzen AI PC. It supports the following execution modes:

- Hybrid execution mode: This mode uses both the NPU and iGPU to achieve the best TTFT and TPS during the prefill and decode phases.
- NPU-only execution mode: This mode uses the NPU exclusively for both the prefill and decode phases.

Supported Configurations

The Ryzen AI OGA flow supports Strix and Krackan Point processors. Phoenix (PHX) and Hawk (HPT) processors are not supported.

Requirements

- Install NPU Drivers and Ryzen AI MSI installer. See *Installation Instructions* for more details.
- Install GPU device driver: Ensure GPU device driver https://www.amd.com/en/support is installed
- Install Git for Windows (needed to download models from HF): https://git-scm.com/downloads

Pre-optimized Models

AMD provides a set of pre-optimized LLMs ready to be deployed with Ryzen AI Software and the supporting runtime for hybrid and/or NPU only execution.

- Phi-3-mini-4k-instruct
- Phi-3.5-mini-instruct
- Mistral-7B-Instruct-v0.3
- Qwen1.5-7B-Chat
- chatglm3-6b
- Llama-2-7b-hf
- Llama-2-7b-chat-hf
- Llama-3-8B
- Llama-3.1-8B
- Llama-3.2-1B-Instruct
- Llama-3.2-3B-Instruct
- Mistral-7B-Instruct-v0.1
- Mistral-7B-Instruct-v0.2
- Mistral-7B-v0.3
- Llama-3.1-8B-Instruct
- CodeLlama-7b-instruct-g128
- DeepSeek-R1-Distill-Llama-8B
- DeepSeek-R1-Distill-Qwen-1.5B
- DeepSeek-R1-Distill-Qwen-7B
- AMD-OLMo-1B-SFT-DPO
- Qwen2-7B
- Qwen2-1.5B
- gemma-2-2b
- Qwen2.5-1.5B-Instruct
- Qwen2.5-3B-Instruct
- Qwen2.5-7B-Instruct

Hugging Face collection of hybrid models: https://huggingface.co/collections/amd/ryzenai-15-llm-hybrid-models-6859a64b421b5c27e1e53899

Hugging Face collection of NPU models: https://huggingface.co/collections/amd/ryzenai-15-llm-npu-models-6859846d7c13f81298990db0

Compatible OGA APIs

Pre-optimized hybrid or NPU LLMs can be executed using the official OGA C++ and Python APIs. The current release is compatible with OGA version 0.7.0. For detailed documentation and examples, refer to the official OGA repository: https://github.com/microsoft/onnxruntime-genai/tree/rel-0.7.0

LLMs Test Programs

The Ryzen AI installation includes test programs (in C++ and Python) that can be used to run LLMs and understand how to integrate them in your application.

The steps for deploying the pre-optimized models using the sample programs are described in the following sections.

C++ Program

Use the `model_benchmark.exe` executable to test LLMs and identify DLL dependencies for C++ applications.

1. (Optional) Enable Performance Mode

To run LLMs in best performance mode, follow these steps:

- Go to `Windows` → `Settings` → `System` → `Power`, and set the power mode to **Best Performance**.
- Open a terminal and run:

```
cd C:\Windows\System32\AMD
xrt-smi configure --pmode performance
```

2. Activate the Ryzen AI 1.5.0 Conda Environment

Run the following command:

```
conda activate ryzen-ai-1.5.0
```

3. Set Up a Working Directory and Copy Required Files

Create a folder and copy the required files into it:

```
mkdir llm_run
cd llm_run

:: Copy the sample C++ executable
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\LLM\example\model_benchmark.exe" .

:: Copy the sample prompt file
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\LLM\example\amd_genai_prompt.txt" .

:: Copy common DLLs
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime-genai.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\ryzen_mm.dll" .

:: Copy DLLs for Hybrid models (skip if using an NPU-only model)
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\onnx_custom_ops.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\libutf8_validity.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\abseil_dll.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\DirectML.dll" .

:: Copy DLLs for NPU-only models (skip if using a Hybrid model)
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_providers_
↪shared.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_providers_
↪vitisai.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_vitis_ai_
↪custom_ops.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\dyn_dispatch_core.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\xaiengine.dll" .
xcopy /Y "%RYZEN_AI_INSTALLATION_PATH%\deployment\onnxruntime_vitisai_ep.
↪dll" .
```

4. Download a Pre-Optimized Model from Hugging Face

Use Git LFS to download the model:

```
:: Install Git LFS if you haven't already: https://git-lfs.com
git lfs install

:: Clone the model repository
git clone https://huggingface.co/amd/Llama-2-7b-chat-hf-awq-g128-int4-
↪asym-fp16-onnx-hybrid
```

5. Run `model_benchmark.exe`

Run the benchmark using the following command:

```
.\model_benchmark.exe -i <path_to_model_dir> -f <prompt_file> -l <list_of_
↪prompt_lengths>

:: Example:
.\model_benchmark.exe -i Llama-2-7b-chat-hf-awq-g128-int4-asym-fp16-onnx-
↪hybrid -f amd_genai_prompt.txt -l "1024"
```

Python Script

Run sample python script

```
python "%RYZEN_AI_INSTALLATION_PATH%\LLM\example\run_model.py" -m <model_
↪folder> -l <max_length>

:: Example command
python "%RYZEN_AI_INSTALLATION_PATH%\LLM\example\run_model.py" -m "Llama-
↪2-7b-chat-hf-awq-g128-int4-asym-fp16-onnx-hybrid" -l 256
```

Building C++ Applications

A complete example including C++ source and build instructions is available in the RyzenAI-SW repository: https://github.com/amd/RyzenAI-SW/tree/main/example/llm/oga_api

LLM Config Files

Each OGA model folder contains a `genai_config.json` file. This file contains various configuration settings for the model. The `session_option` section is where information about specific runtime dependencies is specified. Within this section, the `custom_ops_library` option sets the path to the `onnx_custom_ops.dll` file for Hybrid models and `onnxruntime_vitis_ai_custom_ops.dll` file for NPU models.

The following sample shows the defaults for the AMD pre-optimized Hybrid OGA LLMs:

```
"session_options": {
    "log_id": "onnxruntime-genai",
    "custom_ops_library": "onnx_custom_ops.dll",
    ...
```

The paths is relative to the folder where the program is run from. The model throws an error if the `onnx_custom_ops.dll` file cannot be found at the specified location. Replacing the relative path with an absolute path to this file allows running the program from any location.

Using Fine-Tuned Models

It is also possible to run fine-tuned versions of the pre-optimized OGA models.

To do this, the fine-tuned models must first be prepared for execution with the OGA Hybrid flow. For instructions on how to do this, refer to the page about *Preparing OGA Models*.

After a fine-tuned model has been prepared for Hybrid execution, it can be deployed by following the steps described previously in this page.

Running LLM via pip install

In addition to the full RyzenAI software stack, we also provide standalone wheel files for the users who prefer using their own environment. To prepare an environment for running the Hybrid and NPU-only LLM independently, perform the following steps:

1. Create a new python environment and activate it.

```
conda create -n <env_name> python=3.10 -y
conda activate <env_name>
```

2. Install onnxruntime-genai wheel file.

```
pip install onnxruntime-genai-directml-ryzenai==0.7.0.2.1 --extra-index-
↪url=https://pypi.amd.com/simple
```

3. Navigate to your working directory and download the desired Hybrid/NPU model

```
cd working_directory
git clone <link_to_model>
```

4. Copy the required DLLs from the current environment folder.

```
:: Copy DLLs for Hybrid models (skip if using an NPU-only model)
xcopy "%CONDA_PREFIX%\Lib\site-packages\onnxruntime_genai\onnx_custom_ops.
↪dll" .
xcopy "%CONDA_PREFIX%\Lib\site-packages\onnxruntime_genai\libutf8_
↪validity.dll" .
xcopy "%CONDA_PREFIX%\Lib\site-packages\onnxruntime_genai\abseil_dll.dll"␣
↪.

:: Copy DLLs for NPU-only models (skip if using a Hybrid model)
xcopy "%CONDA_PREFIX%\Lib\site-packages\onnxruntime\capi\onnxruntime_
↪vitis_ai_custom_ops.dll" .
xcopy "%CONDA_PREFIX%\Lib\site-packages\onnxruntime\capi\dyn_dispatch_
↪core.dll" .
xcopy "%CONDA_PREFIX%\Lib\site-packages\onnxruntime\capi\xaiengine.dll" .
```

5. Run the Hybrid or NPU model.

2.2.11 Preparing OGA Models

This section describes the process for preparing LLMs for deployment on a Ryzen AI PC using the hybrid or NPU-only execution mode. Currently, the flow supports only fine-tuned versions of the models already supported (as listed in *OnnxRuntime GenAI (OGA) Flow* page). For example, fine-tuned versions of Llama2 or Llama3 can be used. However, different model families with architectures not supported by the hybrid flow cannot be used.

Preparing a LLM for deployment on a Ryzen AI PC involves 2 steps:

1. **Quantization**: The pretrained model is quantized to reduce memory footprint and better map to compute resources in the hardware accelerators
2. **Postprocessing**: During the postprocessing the model is exported to OGA followed by NPU-only or Hybrid execution mode specific postprocess to obtain the final deployable model.

Quantization

Prerequisites

Linux machine with AMD (e.g., AMD Instinct MI Series) or Nvidia GPUs

Setup

1. Create and activate Conda Environment

```
conda create --name <conda_env_name> python=3.11
conda activate <conda_env_name>
```

2. If Using AMD GPUs, update PyTorch to use ROCm

```
pip3 install torch torchvision torchaudio --index-url https://download.
↪pytorch.org/whl/rocm6.1
python -c "import torch; print(torch.cuda.is_available())" #␣
↪Must return `True`
```

3. Download `AMD Quark 0.9` and unzip the archive
4. Install Quark:

```
cd <extracted amd quark-version>
pip install amd_quark-<version>+<>.whl
```

5. Install other dependencies

```
pip install datasets
pip install transformers
pip install accelerate
pip install evaluate
```

Some models may require a specific version of `transformers`. For example, ChatGLM3 requires version 4.44.0.

Generate Quantized Model

Use following command to run Quantization. In a GPU equipped Linux machine the quantization can take about 30-60 minutes.

```
cd examples/torch/language_modeling/llm_ptq/

python quantize_quark.py \
     --model_dir "meta-llama/Llama-2-7b-chat-hf"  \
     --output_dir <quantized safetensor output dir>  \
     --quant_scheme w_uint4_per_group_asym \
     --num_calib_data 128 \
     --quant_algo awq \
     --dataset pileval_for_awq_benchmark \
     --model_export hf_format \
     --data_type <datatype> \
     --exclude_layers
```

- For a full-precision pretrained model, to generate NPU-only LLM use `--datatype float32`
- For a full-precision pretrained model, to generate Hybrid LLM use `--datatype float16`
- For a BF16 pretrained model, use `--data_type bfloat16`.

The quantized model is generated in the <quantized safetensor output dir> folder.

Postprocessing

Copy the quantized model to the Windows PC with Ryzen AI installed, activate the Ryzen AI Conda environment, and execute `model_generate` command to generate the final model.

Generate the final model for Hybrid execution mode:

```
conda activate ryzen-ai-<version>

model_generate --hybrid <output_dir> <quantized_model_path>
```

Generate the final model for NPU execution mode:

```
conda activate ryzen-ai-<version>

model_generate --npu <output_dir> <quantized_model_path>
```

Known Issue: In the current version, Mistral-7B-Instruct-v0.1 has a known issue during OGA model conversion in the postprocessing stage.

New in 1.5.1:

In Release 1.5.1, a new option has been added to generate a prefill fused version of the Hybrid Model. Currently, it is tested for *Phi-3.5-mini-instruct*, *Llama-2-7b-chat-hf*, and *Llama-3.1-8B-Instruct*.

```
conda activate ryzen-ai-<version>

#For Phi-3.5-mini-instruct/Llama-2-7b-chat-hf
model_generate --hybrid <output_dir> <quantized_model_path> --optimize␣
↪prefill --mode bfp16

#For Llama-3.1-8B-Instruct
model_generate --hybrid <output_dir> <input_quantized_model_path> --
↪optimize prefill_llama3 --mode bfp16
```

After the model is generated, locate the `genai_config.json` file inside the model folder. Edit it as follows:

1. Set `"custom_ops_library"` to `"C:\\Program Files\\RyzenAI\\<release version>\\deployment\\onnx_custom_ops.dll"`
2. Delete `"compile_fusion_rt"` entry from `"amd_options"`
3. Set `dd_cache` to `<output_dir>\\.cache`, for example `"dd_cache": "C:\\Users\\user\\<generated model folder>\\.cache"`
4. For `Phi-3.5-mini-instruct`, `Llama-2-7b-chat-hf` model
   - Set `"hybrid_opt_disable_npu_ops": "1"` inside `"amd_options"`.
   - Set `"fusion_opt_io_bind_kv_cache": "1"` inside `"amd_options"`.
   - Set `"flattened_kv": true` inside `"search"`.

2.2.12 DirectML Flow

Prerequisites

- DirectX12 capable Windows OS (Windows 11 recommended)
- Latest AMD GPU device driver installed

- Microsoft Olive for model conversion and optimization
- Latest ONNX Runtime DirectML EP

You can ensure GPU driver and DirectX version from `Windows Task Manager` -> `Performance` -> `GPU`

Running models on Ryzen AI GPU

Running models on the Ryzen AI GPU is accomplished in two simple steps:

Model Conversion and Optimization: After the model is trained, Microsoft Olive Optimizer can be used to convert the model to ONNX and optimize it for optimal target execution.

For additional information, refer to the Microsoft Olive Documentation

Deployment: Once the model is in the ONNX format, the ONNX Runtime DirectML EP (`DmlExecutionProvider`) is used to run the model on the AMD Ryzen AI GPU.

For additional information, refer to the ONNX Runtime documentation for the DirectML Execution Provider

Examples

- Optimizing and running ResNet on Ryzen AI GPU

Additional Resources

- Article on how AMD and Black Magic Design worked together to accelerate Davinci Resolve Studio workload on AMD hardware:
  - AI Accelerated Video Editing with DaVinci Resolve 18.6 & AMD Radeon Graphics
- Blog posts on using the Ryzen AI Software for various generative AI workloads on GPU:
  - Automatic1111 Stable Diffusion WebUI with DirectML Extension on AMD GPUs
  - Running Optimized Llama2 with Microsoft DirectML on AMD Radeon Graphics
  - AI-Assisted Mobile Workstation Workflows Powered by AMD Ryzen™ AI

2.2.13 NPU Management Interface

Introduction

The `xrt-smi` utility is a command-line interface to monitor and manage the NPU integrated AMD CPUs.

In Window platform, `xrt-smi` is installed in `C:\Windows\System32\AMD` and it can be directly invoked from within the conda environment created by the Ryzen AI Software installer.

The `xrt-smi` utility currently supports three primary commands:

- `examine` - generates reports related to the state of the AI PC and the NPU.
- `validate` - executes sanity tests on the NPU.
- `configure` - manages the performance level of the NPU.

By default, the output of the `xrt-smi examine` and `xrt-smi validate` commands goes to the terminal. It can also be written to file in JSON format as shown below:

```
xrt-smi examine -f JSON -o <path/to/output.json>
```

The utility also support the following options which can be used with any command:

- `--help` - help to use xrt-smi or one of its sub commands
- `--version` - report the version of XRT, driver and firmware
- `--verbose` - turn on verbosity
- `--batch` - enable batch mode (disables escape characters)
- `--force` - when possible, force an operation. Eg - overwrite a file in examine or validate

In Windows, the `xrt-smi` utility requires Microsoft Visual C++ Redistributable (version 2015 to 2022) to be installed.

Overview of Key Commands

Command	Description
examine	system config, device name
examine –report platform	performance mode, power
examine –report aie-partitions	hw contexts
validate –run latency	latency test
validate –run throughput	throughput test
validate –run gemm	INT8 GEMM test TOPS. This is a full array test and it should not be run while another workload is running. **NOTE**: This command is not supported on PHX and HPT NPUs.
configure –pmode <mode>	set performance mode

NOTE: The `examine --report aie-partition` report runtime information. These commands should be used when a model is running on the NPU. You can run these commands in a loop to see live updates of the reported data.

xrt-smi examine

System Information

Reports OS/system information of the AI PC and confirm the presence of the AMD NPU.

```
xrt-smi examine
```

Sample Command Line Output:

```
System Configuration
  OS Name              : Windows NT
  Release              : 26100
  Machine              : x86_64
  CPU Cores            : 20
  Memory               : 32063 MB
  Distribution         : Microsoft Windows 11 Enterprise
  Model                : HP OmniBook Ultra Laptop 14-fd0xxx
  BIOS Vendor          : HP
  BIOS Version         : W81 Ver. 01.01.14

XRT
  Version              : 2.19.0
  Branch               : HEAD
  Hash                 : f62307ddadf65b54acbed420a9f0edc415fefafc
  Hash Date            : 2025-03-12 16:34:48
  NPU Driver Version   : 32.0.203.257
  NPU Firmware Version : 1.0.7.97

Device(s) Present
|BDF             |Name       |
|----------------|-----------|
|[00c4:00:01.1]  |NPU Strix  |
```

Sample Command Line Output in Linux:

```
> xrt-smi examine
  System Configuration
      OS Name              : Linux
      Release              : 6.11.0-26-generic
      Machine              : x86_64
      CPU Cores            : 24
      Memory               : 31440 MB
      Distribution         : Ubuntu 24.04 LTS
      GLIBC                : 2.39
```

(continues on next page)

(continued from previous page)

```
      Model                 : BIRMANPLUS
      BIOS Vendor           : AMD
      BIOS Version          : TXB1001dB

  XRT
      Version               : 2.20.0
      Branch                : master
      Hash                  : 7a277facefecab6c87ac835916021c63d2e395dd
      Hash Date             : 2025-06-16 21:28:35
      amdxdna               : 2.20.0_20250617,␣
↪e7233301f8e4d8d1b1678f3dc3492c826290e314
      NPU Firmware Version : 255.0.1.5

  Device(s) Present
  |BDF              |Name       |
  |-----------------|-----------|
  |[0000:c5:00.1]   |NPU Strix  |
```

Sample JSON Output:

```
{
"schema_version": {
    "schema": "JSON",
    "creation_date": "Tue Mar 18 22:43:38 2025 GMT"
},
"system": {
    "host": {
        "os": {
            "sysname": "Windows NT",
            "release": "26100",
            "machine": "x86_64",
            "distribution": "Microsoft Windows 11 Enterprise",
            "model": "HP OmniBook Ultra Laptop 14-fd0xxx",
            "hostname": "XCOUDAYD02",
            "memory_bytes": "0x7d3f62000",
            "cores": "20",
            "bios_vendor": "HP",
            "bios_version": "W81 Ver. 01.01.14"
        },
        "xrt": {
            "version": "2.19.0",
            "branch": "HEAD",
            "hash": "f62307ddadf65b54acbed420a9f0edc415fefafc",
```

(continues on next page)

(continued from previous page)


```
            "build_date": "2025-03-12 16:34:48",
            "drivers": [
                {
                    "name": "NPU Driver",
                    "version": "32.0.203.257"
                }
            ]
        },
        "devices": [
            {
                "bdf": "00c4:00:01.1",
                "device_class": "Ryzen",
                "name": "NPU Strix",
                "id": "0x0",
                "firmware_version": "1.0.7.97",
                "instance": "mgmt(inst=1)",
                "is_ready": "true"
            }
        ]
    }
  }
}
```


Platform Information

Reports more detailed information about the NPU, such as the performance mode and power consumption.

```
xrt-smi examine --report platform
```

Sample Command Line Output:

```
---------------------------
[00c5:00:01.1] : NPU Strix
---------------------------
Platform
  Name                   : NPU Strix
  Power Mode             : Default

Estimated Power          : 1.277 Watts
```

NOTE: Power reporting is not supported on PHX and HPT NPUs. Power reporting is only available on STX devices and onwards. Report "Estimated Power" is currently unavailable for Linux users.

NPU Partitions

Reports details about the NPU partition and column occupancy on the NPU.

```
xrt-smi examine --report aie-partitions
```

Sample Command Line Output:

```
-------------------------
[00c5:00:01.1] : NPU Strix
-------------------------
AIE Partitions
  Partition Index: 0
    Columns: [0, 1, 2, 3]
    HW Contexts:
      |PID    |Ctx ID  |Status  |Instr BO  |Sub  |Compl  |Migr  |Err ␣
↪|Suspensions  |Prio     |GOPS   |EGOPS  |FPS  |Latency  |
      |-------|--------|--------|----------|-----|-------|------|-----|---
↪----------|---------|-------|-------|-----|---------|
      |20696  |0       |Active  |64 KB     |57   |56     |0     |0    |0 ␣
↪          | Normal  |0      |0      |0    |0        |
```

NPU Context Bindings

Reports details about the columns to NPU HW context binding.

```
xrt-smi examine --report aie-partitions --verbose
```

Sample Command Line Output:

```
Verbose: Enabling Verbosity
Verbose: SubCommand: examine

-------------------------
[00c5:00:01.1] : NPU Strix
-------------------------
AIE Partitions
  Partition Index: 0
    Columns: [0, 1, 2, 3]
    HW Contexts:
      |PID    |Ctx ID  |Status  |Instr BO  |Sub  |Compl  |Migr  |Err ␣
↪|Prio    |GOPS  |EGOPS  |FPS  |Latency  |
      |-------|--------|--------|----------|-----|-------|------|-----|---
↪-----|------|-------|-----|---------|
      |20696  |0       |Active  |64 KB     |57   |56     |0     |0   ␣
```

(continues on next page)

(continued from previous page)

```
↪|Normal  |0     |0      |0    |0        |

AIE Columns
  |Column  ||HW Context Slot  |
  |--------||-----------------|
  |0       ||[1]              |
  |1       ||[1]              |
  |2       ||[1]              |
  |3       ||[1]              |
```

xrt-smi validate

Executing a Sanity Check on the NPU

Runs a set of built-in NPU sanity tests which includes latency, throughput, and gemm.

Note: All tests are run in performance mode.

- `latency` - this test executes a no-op control code and measures the end-to-end latency on all columns
- `throughput` - this test loops back the input data from DDR through a MM2S Shim DMA channel back to DDR through a S2MM Shim DMA channel. The data movement within the AIE array follows the lowest latency path i.e. movement is restricted to just the Shim tile.
- `gemm` - An INT8 GeMM kernel is deployed on all 32 cores by the application. Each core is storing cycle count in the core data memory. The cycle count is read by the firmware. The TOPS application uses the "XBUTIL" tool to capture the IPUHCLK while the workload runs. Once all cores are executed, the cycle count from all cores will be synced back to the host. Finally, the application uses IPUHCLK, core cycle count, and GeMM kernel size to calculate the TOPS. This is a full array test and it should not be run while another workload is running. **NOTE**: This command is not supported on PHX and HPT NPUs.
- `all` - All applicable validate tests will be executed (default)

```
xrt-smi validate --run all
```

NOTE: Some sanity checks may fail if other applications (for example MEP, Microsoft Experience Package) are also using the NPU.

Sample Command Line Output:

```
Validate Device           : [00c4:00:01.1]
   Platform               : NPU Strix
   Power Mode             : Performance
-------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪-----
Test 1 [00c4:00:01.1]     : gemm
   Details                : TOPS: 51.3
   Test Status            : [PASSED]
--------------------------------------------------------------------------------
↪-----
Test 2 [00c4:00:01.1]     : latency
   Details                : Average latency: 84.2 us
   Test Status            : [PASSED]
--------------------------------------------------------------------------------
↪-----
Test 3 [00c4:00:01.1]     : throughput
   Details                : Average throughput: 59891.0 ops
   Test Status            : [PASSED]
--------------------------------------------------------------------------------
↪-----
Validation completed. Please run the command '--verbose' option for more␣
↪details
```

xrt-smi configure

Managing the Performance Level of the NPU

To set the performance level of the NPU, you can choose from the following modes: powersaver, balanced, performance, or default. Use the command below:

```
xrt-smi configure --pmode <default | powersaver | balanced | performance␣
↪| turbo>
```

- `default` - adapts to the Windows Power Mode setting, which can be adjusted under System -> Power & battery -> Power mode. For finer control of the NPU settings, it is recommended to use the xrt-smi mode setting, which overrides the Windows Power mode and ensures optimal results.
- `powersaver` - configures the NPU to prioritize power saving, preserving laptop battery life.
- `balanced` - configures the NPU to provide a compromise between power saving and performance.
- `performance` - configures the NPU to prioritize performance, consuming more power.
- `turbo` - configures the NPU for maximum performance performance, requires AC power to be plugged in otherwise uses `performance` mode.

Example: Setting the NPU to high-performance mode

```
xrt-smi configure --pmode performance
```

To check the current power mode, use the following command:

```
xrt-smi examine --report platform
```

2.2.14 AI Analyzer

AMD AI Analyzer is a tool that supports analysis and visualization of model compilation and inference on Ryzen AI. The primary goal of the tool is to help you better understand how the models are processed by the hardware, and to identify performance bottlenecks that may be present during model inference. Using AI Analyzer, you can visualize graph and operator partitions between the NPU and CPU.

Installation

If you installed the Ryzen AI software using automatic installer, AI Analyzer is already installed in the conda environment.

If you manually installed the software, you need to install the AI Analyzer wheel file in your environment.

```
python -m pip install path\to\RyzenAI\installation\files\aianalyzer-
↪<version>.whl
```

Enabling Profiling and Visualization

Profiling and Visualization can be enabled by passing additional provider options to the ONNXRuntime Inference Session. Here is an example:

```
provider_options = [{
            'config_file': 'vaip_config.json',
            'cacheDir': str(cache_dir),
            'cacheKey': 'modelcachekey',
            'ai_analyzer_visualization': True,
            'ai_analyzer_profiling': True,
        }]
session = ort.InferenceSession(model.SerializeToString(),␣
↪providers=providers,
                               provider_options=provider_options)
```

The `ai_analyzer_profiling` flag enables generation of artifacts related to the inference profile. The `ai_analyzer_visualization` flag enables generation of artifacts related to graph partitions and operator fusion. These artifacts are generated as JSON files in the current run directory.

AI Analyzer also supports native ONNX Runtime profiling, which you can use to analyze the parts of the session that run on the CPU. You can enable ONNX Runtime profiling through session options and pass it along with the provider options, as shown here:

```
 # Configure session options for profiling
 sess_options = ort.SessionOptions()
 sess_options.enable_profiling = True

 provider_options = [{
              'config_file': 'vaip_config.json',
              'cacheDir': str(cache_dir),
              'cacheKey': 'modelcachekey',
              'ai_analyzer_visualization': True,
              'ai_analyzer_profiling': True,
          }]

session = ort.InferenceSession(model.SerializeToString(), sess_options,␣
↪providers=providers,
                               provider_options=provider_options)
```

Launching AI Analyzer

After the artifacts are generated, *aianalyzer* can be invoked through the command line as follows:

```
aianalyzer <logdir> <additional options>
```

Positional Arguments

`logdir`: Path to the folder containing generated artifacts

Additional Options

`-v`, `--version`: Show the version info and exit.

`-b ADDR`, `--bind ADDR`: Hostname or IP address on which to listen, default is 'localhost'.

`-p PORT`, `--port PORT`: TCP port on which to listen, default is '8000'.

`-n`, `--no-browser`: Prevents the opening of the default url in the browser.

`-t TOKEN`, `--token TOKEN`: Token used for authenticating first-time connections to the server. The default is to generate a new, random token. Setting to an empty string disables authentication altogether, which is not recommended.

Features

AI Analyzer provides visibility into how your AI model is compiled and executed on Ryzen AI hardware. Its two main use cases are:

1. Analyzing how the model was partitioned and mapped onto Ryzen AI's CPU and NPU accelerator
2. Profiling model performance as it executes inferencing workloads

When launched, the AI Analyzer server scans the folder specified with the logdir argument and detect and load all files relevant to compilation and/or inferencing per the *ai_analyzer_visualization* and *ai_anlayzer_profiling* flags.

You can instruct the AI Analyzer server to either start a browser on the same host or return an URL that you can then load into a browser on any host.

User Interface

AI Analyzer has the following three sections as seen in the left-panel navigator:

1. PARTITIONING - A breakdown of your model was assigned to execute inference across CPU and NPU
2. NPU INSIGHTS - A detailed look at the how your model was optimized for inference execution on NPU
3. PERFORMANCE - A breakdown of inference execution through the model

These sections are described in more detail in the following sections:

PARTITIONING

This section is comprised of two pages: Summary and Graph

Summary

The Summary page gives an overview of how the models operators have been assigned to Ryzen's CPU and NPU along with charts capturing GigaOp (GOP) offloading by operator type .

There is also table titled "CPU Because" that shows the reasons why certain operators were not offloaded to the NPU.

Graph

The graph page shows an interactive diagram of the partitioned ONNX model, showing graphically how the layers are assigned to the Ryzen hardware.

Toolbar

- You can choose to show/hide individual NPU partitions, if any, with the **Filter by Partition** button

- You can show or hide a panel that displays properties for selected objects through the **Show Properties** toggle button
- You can show or hide the model table through the **Show Table** toggle button.
- Settings
  - Show Processor separates operators that run on CPU and NPU respectively
  - Show Partition separates operators running on the NPU by their respective NPU partition, if any
  - Show Instance Name displays the full hierarchical name for the operators in the ONNX model

All objects in the graph have properties that can be viewed to the right of the graph.

Model Table

This table following the graph lists all objects in the partitioned ONNX model:

- Processor (NPU or CPU)
- Function (Layer)
- Operator
- Ports
- NPU Partitions

NPU INSIGHTS

This section is comprised of three pages: Summary, Original Graph, and Optimized Graph.

Summary

The Summary page gives an overview of how your model was mapped to the AMD Ryzen NPU. Charts are displayed showing statistics on the number of operators and total GMACs that have been mapped to the NPU (and if necessary, back to CPU via the *Failsafe CPU* mechanism). The statistics are shown per operator type and NPU partition.

Original Graph

This is an interactive graph representing your model, lowered to supported NPU primitive operators and divided into partitions if necessary. As with the PARTITIONING graph, a companion table lists all model elements and supports cross-probing with the graph view. The objects in both the graph and the table also cross-probe with the PARTITIONING graph.

Toolbar

You can choose to show/hide individual NPU partitions, if any, with the **Filter by Partition** button A panel that displays properties for selected objects can be shown or hidden using the **Show Properties** toggle button A code viewer showing the MLIR source code with cross-probing can

be shown/hidden through the **Show Code View** button The following table can be shown and hidden using the **Show Table** toggle button. Display options for the graph can be accessed with the **Settings** button

Optimized Graph

This page shows the final model that is mapped to the NPU after all transformations and optimizations such as fusion and chaining. It also reports the operators that had to be moved back to the CPU through the *Failsafe CPU* mechanism. As usual, there is a companion table below that contains all of the graph's elements, and cross-selection is supported to and from the PARTITIONING graph and the Original Graph.

Toolbar

You can choose to show/hide individual NPU partitions, if any, with the **Filter by Partition** button A panel that displays properties for selected objects can be shown or hidden using the **Show Properties** toggle button The following table can be shown and hidden using the **Show Table** toggle button. Display options for the graph can be accessed with the **Settings** button

PERFORMANCE

Use this section to view the performance of your model on RyzenAI when running one or more inferences. It is comprised of two pages: Summary and Timeline.

Summary

The performance summary page displays several overall statistics for the inference(s), along with charts that break down operator runtime by operator. When the ONNX Runtime profiler is enabled, the total inference time, including layers executed on the CPU, is shown. When NPU profiling is enabled using the *ai_analyzer_profiling* flag, additional NPU-specific statistics are displayed, including GOP and MAC efficiency, as well as a chart showing runtime per NPU operator type.

The clock frequency field shows the assumed NPU clock frequency, but it is editable. When the frequency is changed, all timestamp data—collected as clock cycles but displayed in time units—is adjusted accordingly.

Timeline

The Performance timeline shows a layer-by-layer breakdown of your model's execution. The upper section is a graphical depiction of layer execution across a timeline, while the lower section shows the same information in tabular format. It is important to note that the Timeline page shows one inference at a time, so if you have captured profiling data for two or more inferences, you can choose which one to display with the **Inferences** chooser.

Within each inference, you can examine the overall model execution or the detailed NPU execution data by using the **Partition** chooser.

Toolbar

A panel that displays properties for selected objects can be shown or hidden using the **Show Properties** toggle button The following table can be shown and hidden using the **Show Table** toggle

button. The graphical timeline can be downloaded to SVG using the **Export to SVG** button

2.2.15 Stable Diffusion Demo

Ryzen AI 1.5 provides preview demos of Stable Diffusion image-generation pipelines. The demos cover Image-to-Image and Text-to-Image using SD 1.5, SD 2.1-base, SD-Turbo, SDXL-Turbo and SD 3.0.

The models for SD 1.5, SD 2.1-base, SD-Turbo, SDXL-Turbo are available for public download. The SD 3.0 models are only available to confirmed Stability AI licensees.

NOTE: Preview features are features which are still undergoing some optimization and fine-tuning. These features are not in their final form and may change as we continue to work in order to mature them into full-fledged features.

Installation Steps

1. Ensure the latest version of Ryzen AI and NPU drivers are installed. See *Installation Instructions*.
2. Copy the GenAI-SD folder from the RyzenAI installation tree to your working area, and then go to the copied folder. For instance:

```
xcopy /I /E "C:\Program Files\RyzenAI\1.5.0\GenAI-SD" C:\Temp\GenAI-SD
cd C:\Temp\GenAI-SD
```

3. Create a Conda environment for the Stable Diffusion demo packages:

```
conda update -n base -c defaults conda
conda env create --file=env.yaml
```

4. Download the Stable Diffusion models:
    - `GenAI-SD-models-v0613-v0711.zip`
    - `GenAI-SDXL-turbo-models-v0613-v0711.zip`
5. Extract the downloaded zip files and copy the models in the `GenAI-SD\models` folder. After installing all the models, the `GenAI-SD\models` folder should contain the following subfolders:
    - sd15_controlnet
    - sd_15
    - sd_21_base
    - sd_turbo
    - sdxl_turbo

Running the Demos

Activate the conda environment:

```
conda activate ryzenai-stable-diffusion
```

Optionally, set the NPU to high performance mode to maximize performance:

```
xrt-smi configure --pmode performance
```

Refer to the documentation on *xrt-smi configure* for additional information.

Image-to-Image with ControlNet

The image-to-image demo generates images based on a prompt and a control image for a Canny ControlNet. This demo supports SD 1.5 (512x512).

To run the demo, navigate to the `GenAI-SD\test` directory and run the following command:

```
python .\run_sd15_controlnet.py
```

The demo script uses a predefined prompt and `ref\control.png` as the control image. The output image and control image are saved in the `generated_images` folder.

The control image can be modified and custom prompts can be provided with the `--prompt` option. For instance:

```
python run_sd15_controlnet.py --prompt "A red bird on a grey sky"
```

Text-to-Image

The text-to-image generates images based on text prompts. This demo supports SD 1.5 (512x512), SD 2.1-base (768x768), SD-Turbo (512x512) and SDXL-Turbo (512x512).

To run the demo, navigate to the `GenAI-SD\test` directory and run the following commands to run with each of the supported models:

```
python run_sd.py    --model_id 'stable-diffusion-v1-5/stable-diffusion-v1-
↪5' --model_path ..\models\sd_15
python run_sd.py    --model_id 'stabilityai/stable-diffusion-2-1-base' --
↪model_path ..\models\sd_21_base
python run_sd.py    --model_id 'stabilityai/sd-turbo' --model_path ..\
↪models\sd_turbo
python run_sd_xl.py --model_id 'stabilityai/sdxl-turbo' --model_path ..\
↪models\sdxl_turbo
```

The demo script uses a predefined prompt for each of the models. The output images are saved in the `generated_images` folder.

Custom prompts can be provided with the `--prompt` option. For instance:

```
python run_sd.py --model_id 'stabilityai/stable-diffusion-2-1-base' --
↪model_path ..\models\sd_21_base  --prompt "A bouquet of roses,␣
↪impressionist style"
```

2.2.16 Ryzen AI CVML library

The Ryzen AI CVML libraries build on top of the Ryzen AI drivers and execution infrastructure to provide powerful AI capabilities to C++ applications without having to worry about training specific AI models and integrating them to the Ryzen AI framework.

Each Ryzen AI CVML library feature offers a simple C++ application programming interface (API) that can be easily incorporated into existing applications. The following AI features are currently available,

- **Depth Estimation**: Generates a depth map to assess relative distances within a two-dimensional image.
- **Face Detection**: Identifies and locates faces within an image.
- **Face Mesh**: Constructs a mesh overlay of landmarks for a specified facial image.

The Ryzen AI CVML library is distributed through the RyzenAI-SW Github repository: https://github.com/amd/RyzenAI-SW/tree/main/Ryzen-AI-CVML-Library

Building sample applications

This section describes the steps to build Ryzen AI CVML library sample applications. Before starting, ensure that the following prerequisites are available in the build environment,

- CMake, version 3.18 or newer
- C++ compilation toolchain. On Windows, this may be Visual Studio's "Desktop development with C++" build tools, or a comparable C++ toolchain
- OpenCV, version 4.11 or newer

Navigate to the folder containing Ryzen AI samples

Download the Ryzen AI CVML sources, and go to the 'samples' sub-folder of the library.

On Windows,

```
git clone https://github.com/amd/RyzenAI-SW.git -b main --depth-1
cd RyzenAI-SW\Ryzen-AI-CVML-Library\samples
```

On Linux,

```
git clone https://github.com/amd/RyzenAI-SW.git -b main --depth-1
cd RyzenAI-SW/Ryzen-AI-CVML-Library/samples
```

OpenCV libraries

Ryzen AI CVML library samples make use of OpenCV, so set an environment variable to let the build scripts know where to find OpenCV.

On Windows,

```
set OPENCV_INSTALL_ROOT=<location of OpenCV libraries>
```

On Linux,

```
export OPENCV_INSTALL_ROOT=<location of OpenCV libraries>
```

Build Instructions

Create a build folder and use CMAKE to build the sample(s).

On Windows,

```
mkdir build
cmake -S %CD% -B %CD%\build -DOPENCV_INSTALL_ROOT=%OPENCV_INSTALL_ROOT%
cmake --build %CD%\build --config Release
```

On Linux,

```
mkdir build
cmake -S $PWD -B $PWD/build -DOPENCV_INSTALL_ROOT=$OPENCV_INSTALL_ROOT
cmake --build $PWD/build --config Release
```

The compiled sample application(s) will be placed in the various build<application>Release folder(s) under the 'samples' folder (or build/<application>/Release for Linux).

Running sample applications

This section describes how to execute Ryzen AI CVML library sample applications.

Update the console and/or system PATH

Ryzen AI CVML library applications need to be able to find the library files.

On Windows, update the PATH environment variable for both the Ryzen AI CVML library location and OpenCV

```
set PATH=%PATH%;<location of Ryzen AI CVML library package>\windows
set PATH=%PATH%;%OPENCV_INSTALL_ROOT%\x64\vc16\bin
```

On Linux, update LD_LIBRARY_PATH for the Ryzen AI CVML library location, OpenCV library location and NPU driver location,

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<location of Ryzen AI CVML␣
↪library package>/linux
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OPENCV_INSTALL_ROOT/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/xilinx/xrt/lib
```

Adjust the aforementioned commands to match the actual location of Ryzen AI and OpenCV libraries, respectively.

Select an input source (image or video)

Ryzen AI CVML library samples can accept a variety of image and video input formats, or even open the default camera on the system if "0" is specified as an input.

In this example, a publicly available video file is used for the application's input. The following command downloads a video file and saves it locally as 'dancing.mp4'

```
curl -o dancing.mp4 https://videos.pexels.com/video-files/4540332/4540332-
↪hd_1920_1080_25fps.mp4
```

Execute the sample application

Finally, the previously built sample application may be executed with the selected input source.

On Windows,

```
build\cvml-sample-depth-estimation\Release\cvml-sample-depth-estimation.
↪exe -i dancing.mp4
```

On Linux,

```
build/cvml-sample-depth-estimation/Release/cvml-sample-depth-estimation.
↪exe -i dancing.mp4
```

2.2.17 Licensing Information

Ryzen AI is released by Advanced Micro Devices, Inc. (AMD) and is subject to the licensing terms listed below. Some components may include third-party software that is subject to additional licenses. Review the following links for more information:

- [AMD End User License Agreement](https://account.amd.com/content/dam/account/en/licenses/download/amd-end-user-license-agreement.pdf)
- [Third Party End User License Agreement](https://account.amd.com/content/dam/account/en/licenses/download/ryzenai-1.5-ga-win64-tpn-license.pdf)

INDEX