PyTorch-ZenDNN User Guide

Publication # **57608**	Revision # **4.0**
Issue Date **January 2023**	

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

Trademarks

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Dolby is a trademark of Dolby Laboratories.

ENERGY STAR is a registered trademark of the U.S. Environmental Protection Agency.

HDMI is a trademark of HDMI Licensing, LLC.

HyperTransport is a licensed trademark of the HyperTransport Technology Consortium.

Microsoft, Windows, Windows Vista, and DirectX are registered trademarks of Microsoft Corporation.

MMX is a trademark of Intel Corporation.

OpenCL is a trademark of Apple Inc. used by permission by Khronos.

PCIe is a registered trademark of PCI-Special Interest Group (PCI-SIG).

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

Dolby Laboratories, Inc.

Manufactured under license from Dolby Laboratories.

Rovi Corporation

This device is protected by U.S. patents and other intellectual property rights. The use of Rovi Corporation's copy protection technology in the device must be authorized by Rovi Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by Rovi Corporation.

Reverse engineering or disassembly is prohibited.

USE OF THIS PRODUCT IN ANY MANNER THAT COMPLIES WITH THE MPEG-2 STANDARD IS EXPRESSLY PROHIBITED WITHOUT A LICENSE UNDER APPLICABLE PATENTS IN THE MPEG-2 PATENT PORTFOLIO, WHICH LICENSE IS AVAILABLE FROM MPEG LA, L.L.C., 6312 S. FIDDLERS GREEN CIRCLE, SUITE 400E, GREENWOOD VILLAGE, COLORADO 80111.

Contents

Revision History **6**
Chapter 1 Installing ZenDNN with PyTorch **7**
1.1 Binary Release Setup 7
1.1.1 Conda 7
Chapter 2 Directory Structure **9**
Chapter 3 High-level Overview **10**
Chapter 4 PyTorch CNN Benchmarks **11**
Chapter 5 PyTorch v1.12 . . . . . **14**
Chapter 6 Environment Variables **15**
Chapter 7 Tuning Guidelines **17**
7.1 System 17
7.2 Environment Variables 17
7.3 Thread Wait Policy 18
7.4 Thread Affinity 18
7.5 Non-uniform Memory Access 19
7.5.1 numactl 19
7.6 Transparent Huge Pages 19
7.7 Memory Allocators 20
7.7.1 JEMalloc 20
7.7.2 Usage 20
Chapter 8 License **22**
Chapter 9 Technical Support **23**

List of Figures

Figure 1. ZenDNN Library. 10

List of Tables

Table 1. ZenDNN Environment Variables-Generic 15
Table 2. ZenDNN Environment Variables-Optimization 16
Table 3. System Specification 17

Revision History

Date	Revision	Description
January 2023	4.0	• Updated supported PyTorch version. • Added sections 1.1.1 and 7.3 through 7.7.
June 2022	3.3	Updated supported PyTorch version.
December 2021	3.2	Initial version.

Chapter 1 Installing ZenDNN with PyTorch

Note: *Refer to the ZenDNN v4.0 User Guide before starting the installation.*

In this release, we are providing ZenDNN library support for PyTorch v1.12. This is a baseline release for PyTorch v1.12 with:

- FP32 support
- Only AMD UIF 1.1 INT8 model support
- Limited support for BF16 on AMD UIF 1.1 ResNet50
- INT16 is not supported in this release

1.1 Binary Release Setup

1.1.1 Conda

Complete the following steps to setup Conda:

1. Refer to Anaconda documentation (*https://docs.anaconda.com/anaconda/install/linux/*) to install Anaconda on your system.
2. Create and activate a Conda environment which will house all the PyTorch-ZenDNN specific installations:

   ```
   conda create -n pt-v1.12-zendnn-v4.0-rel-env python=3.8

   conda activate pt-v1.12-zendnn-v4.0-rel-env
   ```

   Ensure that you install the PyTorch-ZenDNN package corresponding to the Python version with which you created the Conda environment.

   Note: *If there is any conda environment named pt-1.12-zendnn-v4.0-rel-env already present, delete the conda environment pt-1.12-zendnn-v4.0-rel-env (using command conda remove --name pt-1.12-zendnn-v4.0-rel-env --all) before running scripts/ PT_ZenDNN_setup_release.sh.*
3. It is recommended to use the naming convention:

   ```
   pt-v1.12-zendnn-v4.0-rel-env
   ```

4. Install all the necessary dependencies:

   ```
   pip install --upgrade typing-extensions

   pip install --upgrade numpy==1.23.2
   ```

   Note: *For binary packages built with Python v3.7, it is recommended to use numpy v1.21.6 (numpy==1.21.6).*

Complete the following steps to install the ZenDNN binary release:

1. Copy the zipped release package to the local system being used. The name of the release package will be similar to *PT_v1.12_ZenDNN_v4.0_Python_v3.8.zip*.
2. Execute the following commands:
   a. `unzip PT_v1.12_ZenDNN_v4.0_Python_v3.8.zip`
   b. `cd PT_v1.12_ZenDNN_v4.0_Python_v*/`
   c. `source scripts/PT_ZenDNN_setup_release.sh`

      This installs the PyTorch wheel package provided in the zip file.

      Note: *Ensure that it is sourced only from the folder PT_v1.12_ZenDNN_v4.0_Python_v*/.*
   d. To run the benchmarks with different CNN models at the PyTorch level, refer the section "PyTorch CNN Benchmarks" on page 11.

The release binaries for PyTorch v1.12 are now compiled with manylinux2014 and they provide compatibility with some older Linux distributions.

The release binaries are tested with the recent Linux distributions such as:

- Ubuntu 20.04 and later
- RHEL 9.0 and later

C++ Interface will work on operating systems (with glibc version 2.31 or later) such as:

- Ubuntu 20.04 and later
- RHEL 9.0 and later

Chapter 2 Directory Structure

The release folder consists of a PyTorch wheel (.whl), LICENSE and THIRD-PARTY-PROGRAMS files, and the following directories:

- *scripts* contains scripts to install the wheel file and run benchmarks

Chapter 3 High-level Overview

The following is a high-level block diagram for the ZenDNN library, which uses the AOCL-BLIS library internally:

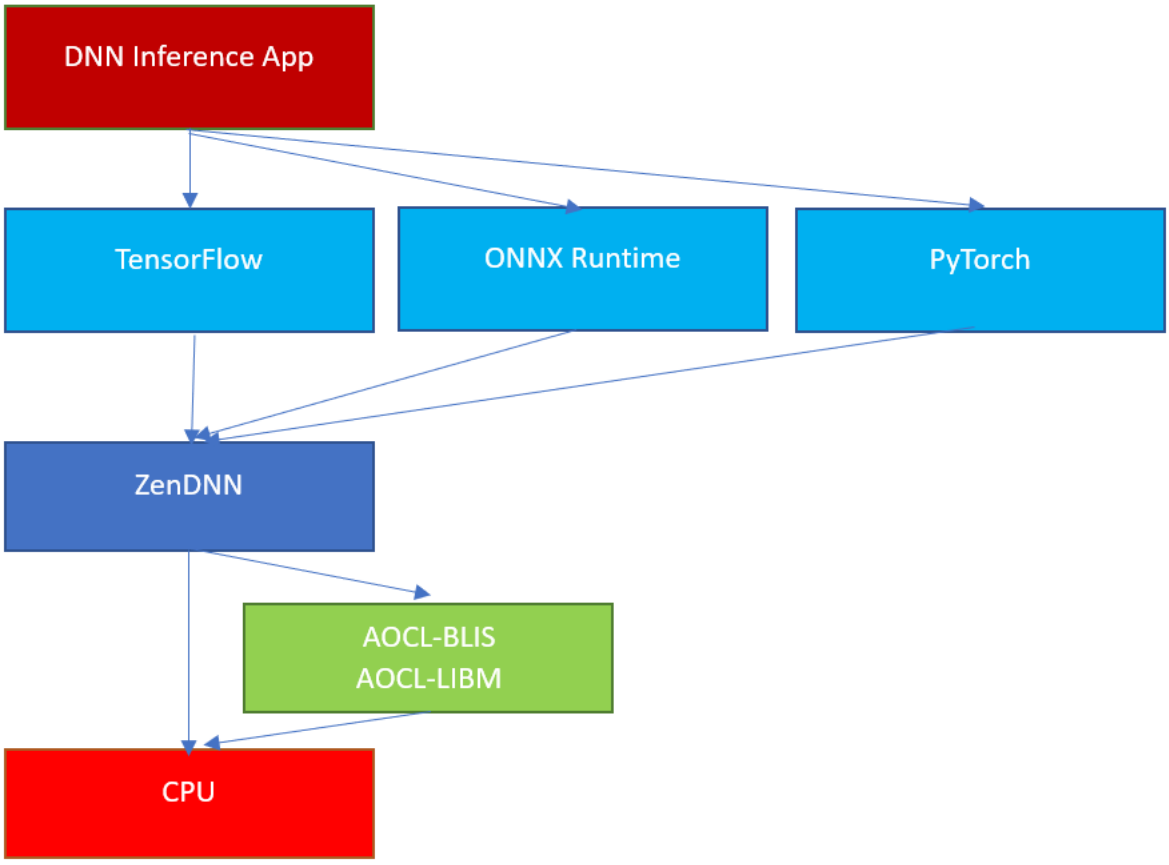

Figure 1. ZenDNN Library

In the current release, ZenDNN is integrated with TensorFlow, ONNXRuntime, and PyTorch.

Chapter 4 PyTorch CNN Benchmarks

The benchmark scripts provide performance benchmarking at the PyTorch level. It prints the latency and throughput results for the following torchvision supported models:

ResNet50, ResNet152, GoogLeNet, and VGG11

To install JEMalloc, complete the following steps:

1. Follow the steps on JEMalloc installation (*https://github.com/jemalloc/jemalloc/blob/dev/INSTALL.md*).
2. Export following environment variables:

```
export LD_PRELOAD=<Installation path>lib/libjemalloc.so
export MALLOC_CONF="oversize_threshold:1,background_thread:true,meta-
data_thp:auto,dirty_decay_ms:-1,muzzy_decay_ms:-1"
```

To install torchvision, execute the following command:

```
pip install torchvision==0.13.0+cpu --extra-index-url https://download.pytorch.org/whl/cpu
```

For latency, execute the following commands:

1. `cd PT_v1.12_ZenDNN_v4.0_Python_v*/`
2. `source scripts/zendnn_PT_env_setup.sh`
3. `conda activate pt-v1.12-zendnn-v4.0-rel-env`
4. `bash scripts/pt_cnn_benchmarks_latency.sh`

For throughput, execute the following commands:

1. `cd PT_v1.12_ZenDNN_v4.0_Python_v*/`
2. `source scripts/zendnn_PT_env_setup.sh`
3. `conda activate pt-v1.12-zendnn-v4.0-rel-env`
4. `bash scripts/pt_cnn_benchmarks_throughput.sh`

To run individual models rather than the entire suite, execute the following commands:

```
cd $ZENDNN_PARENT_FOLDER/scripts/
numactl --cpunodebind=<NPS> --interleave=<NPS> python pt_cnn_benchmarks.py --arch
<model_name> --batch_size $BATCH_SIZE --iterations $NUM_OF_BATCHES --warmups %WARMUP_-
SIZE
```

Replace `<NPS>` with the following based on your number of NUMA nodes. Execute the command `lscpu` to identify the number of NUMA nodes for your machine:

- If you have 1 NUMA node, replace `<NPS>` with 0

- If you have 2 NUMA nodes, replace `<NPS>` with 0-1
- If you have 4 NUMA nodes, replace `<NPS>` with 0-3

Replace `<model_name>` with one of the following options:

- For ResNet50, replace `<model_name>` with resnet50
- For ResNet152, replace `<model_name>` with resnet152
- For GoogLeNet, replace `<model_name>` with googlenet
- For VGG11, replace `<model_name>` with vgg11

While executing the commands, make a note of the following:

- For optimal settings, refer to the Tuning Guidelines section. Current setting refers to 96C, 2P, SMT=ON configuration.
- If the following warning is displayed on the terminal, it can be ignored. During the environment setup, there is an optional script to gather information about hardware, OS, Kernel, and BIOS and it requires a few utilities (lscpu, lstopo-no-graphics, dmidecode, and so on) to be present. If these utilities are not present, you may encounter the following warning:

```
scripts/gather_hw_os_kernel_bios_info.sh
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: _HW_LSTOPO_NUM_L2CACHE/_HW_LSTOPO_PACKAGES: division by 0 (error token is
"_HW_LSTOPO_PACKAGES")
bash: _HW_LSTOPO_NUM_L3CACHE/_HW_LSTOPO_PACKAGES: division by 0 (error token is
"_HW_LSTOPO_PACKAGES")
sudo: dmidecode: command not found
sudo: dmidecode: command not found
sudo: dmidecode: command not found
```

- If a warning similar to the following appears during benchmark runs, configure your GOMP_CPU_AFFINITY setting to match the number of CPU cores supported by your machine:

```
OMP: Warning #181: OMP_PROC_BIND: ignored because GOMP_CPU_AFFINITY is defined

OMP: Warning #123: Ignoring invalid OS proc ID 48

OMP: Warning #123: Ignoring invalid OS proc ID 49

.

.

.

OMP: Warning #123: Ignoring invalid OS proc ID 63
```

  For example, if your CPU has 24 cores, your GOMP_CPU_AFFINITY should be set as "export GOMP_CPU_AFFINITY=0-23".

- If a warnings similar to the following appear during the benchmark runs, they may be ignored:

```
“../site-packages/torchvision/models/googlenet.py:212: UserWarning: Scripted Goo-
gleNet always returns GoogleNetOutputs Tuplewarnings.warn("Scripted GoogleNet always
returns GoogleNetOutputs Tuple")”
```

```
"UserWarning: Arguments other than a weight enum or `None` for 'weights' are deprecated
since 0.13 and will be removed in 0.15" and "UserWarning: The parameter 'pretrained' is
deprecated since 0.13 and will be removed in 0.15, please use 'weights' instead."
```

Chapter 5 PyTorch v1.12

In this release of ZenDNN:

- ZenDNN library is supported for PyTorch v1.12. This is a baseline release for PyTorch v1.12.
- AMD Unified Inference Frontend (UIF) optimized models are supported. For the model details, refer to the UIF documentation.
- PyTorch v1.12 wheel file is compiled with GCC v9.3.1.
- PyTorch v1.12 is expected to deliver similar or better performance as compared to PyTorch v1.11.

Chapter 6 Environment Variables

ZenDNN uses the following environment variables to setup paths and control logs:

Table 1. ZenDNN Environment Variables-Generic

Environment Variable	Default Value/User Defined Value
ZENDNN_LOG_OPTS	ALL: 0
ZENDNN_PARENT_FOLDER	Path to unzipped release folder
ZENDNN_PRIMITIVE_CACHE_CAPACITY	The default value is set to 1024, you can modify it as required.
OMP_DYNAMIC	FALSE

The following is a list of environment variables to tune performance:

Table 2. ZenDNN Environment Variables-Optimization

Environment Variable	Default Value/User Defined Value
OMP_NUM_THREADS	The default value is set to 96. You can set it as per the number of cores in the user system[a].
OMP_WAIT_POLICY	ACTIVE
OMP_PROC_BIND	FALSE
GOMP_CPU_AFFINITY	Set it as per the number of cores in the system being used. For example, use 0-95 for 96-core servers.
ZENDNN_GEMM_ALGO	The default value is 3. You can modify it to any of the following: • 0 = Auto • 1 = BLIS path • 2 = partial-BLIS • 3 = ZenDNN jit path • 4 = ZenDNN partial jit ***Note: Auto is an experimental feature.***

a. You must set these environment variables explicitly.

Note: *There are a few other environment variables that are initialized by the setup script, however these are not applicable for the binary release setup.*

When source *scripts/zendnn_PT_env_setup.sh* is invoked, the script initializes all the environment variables except the one(s) which must be set manually. The environment variable **ZENDNN_PARENT_FOLDER** is initialized relative to the unzipped release folder. To ensure that the paths are initialized correctly, it is important that the script is invoked from the unzipped release folder.

Chapter 7 Tuning Guidelines

The hardware configuration, OS, Kernel, and BIOS settings play an important role in performance. The details for the environment variables used on a 4th Gen AMD EPYC™ server to get the best performance numbers are as follows:

7.1 System

A system with the following specifications has been used:

Table 3. System Specification

Model name	4th Gen AMD EPYC™ 9654P 96-Core Processor
CPU MHz	Up to 3.7 GHz
No of Cores	96
1P/2P	1
SMT: Thread(s) per Core	2
Mem-Dims	12x64 GB

7.2 Environment Variables

The following environment variables have been used:

ZENDNN_LOG_OPTS=ALL:0

Note: *When ZenDNN logs are enabled, you will encounter the following warning which can be ignored:*

WARNING: Current PyTorch is not built with ZenDNN support which can be ignored

OMP_NUM_THREADS=96

OMP_WAIT_POLICY=ACTIVE

OMP_PROC_BIND=FALSE

OMP_DYNAMIC=FALSE

ZENDNN_GEMM_ALGO=3

ZENDNN_GIT_ROOT=/home/<user_id>/my_work/

ZENDNN_PARENT_FOLDER=/home/<user_id>/my_work

ZENDNN_PRIMITIVE_CACHE_CAPACITY=1024

GOMP_CPU_AFFINITY=0-95

As mentioned in "Environment Variables" on page 15, the script *scripts/zendnn_PT_env_setup.sh*, initializes all the environment variables except the one(s) which you must set manually. The environment variables **OMP_NUM_THREADS**, **OMP_WAIT_POLICY**, **OMP_PROC_BIND**, and **GOMP_CPU_AFFINITY** can be used to tune performance. For optimal performance, the **Batch Size** must be a multiple of the total number of cores (used by the threads). On a 4th Gen AMD EPYC^{TM} server (configuration: AMD EPYC^{TM} 9654P 96-Core, 2P, and **SMT=ON**) with the above environment variable values, **OMP_NUM_THREADS=96** and **GOMP_CPU_AFFINITY=0-95** yield the best throughput numbers.

7.3 Thread Wait Policy

OMP_WAIT_POLICY environment variable provides options to the OpenMP runtime library based on the expected behavior of the waiting threads. It can take the abstract values **PASSIVE** and **ACTIVE**. The default value is **ACTIVE**. When **OMP_WAIT_POLICY** is set to **PASSIVE**, the waiting threads will be passive and will not consume the processor cycles. Whereas, setting it to **ACTIVE** will consume processor cycles.

Note: *For ZenDNN stack, setting **OMP_WAIT_POLICY** to **ACTIVE** may give better performance.*

7.4 Thread Affinity

To improve ZenDNN performance, the behavior of OpenMP thread can be guarded precisely with thread affinity settings. A thread affinity defined at start up cannot be modified or changed during runtime of the application. Following are the ways through which you can bind the requested OpenMP threads to the physical CPUs:

GOMP_CPU_AFFINITY environment variable binds threads to the physical CPUs, for example:

export GOMP_CPU_AFFINITY="0 3 1-2 4-15:2"

This command will bind the:

- Initial thread to CPU 0
- Second thread to CPU 3
- Third and fourth threads to CPU 1 and CPU 2 respectively
- Fifth thread to CPU 4
- Sixth through tenth threads to CPUs 6, 8, 10, 12, and 14 respectively

Then, it will start the assigning back from the beginning of the list.

export GOMP_CPU_AFFINITY="0" binds all the threads to CPU 0.

Example:

Following affinity settings should give the same thread bindings:

export GOMP_CPU_AFFINITY=0-95

7.5 Non-uniform Memory Access

7.5.1 numactl

numactl provides options to run processes with specific scheduling and memory placement policy. It can restrict the memory binding and process scheduling to specific CPUs or NUMA nodes:

- `--cpunodebind=nodes`: Restricts the process to specific group of nodes.
- `--physcpubind=cpus`: Restricts the process to specific set of physical CPUs.
- `--membind=nodes`: Allocates the memory from the nodes listed. The allocation fails if there is not enough memory on the listed nodes.
- `--interleave=nodes`: Memory will be allocated in a round robin manner across the specified nodes. When the memory cannot be allocated on the current target node, it will fall back to the other nodes.

Example:

If <pytorch_script> is the application that needs to run on the server, then it can be triggered using `numactl` settings as follows:

```
numactl --cpunodebind=0-3 -interleave=0-3 python <pytorch_script>
```

The `interleave` option of numactl works only when the number nodes allocated for a particular application is more than one. `cpunodebind` and `physcpubind` behave the same way for ZenDNN stack, whereas `interleave` memory allocation performs better than `membind`.

The number of concurrent executions can be increased beyond 4 nodes. The following formula can be used to decide the number of concurrent executions to be triggered at a time:

```
Number Concurrent Executions = Number of Cores Per Socket / Numbers of Cores sharing L3
cache
```

This can also be extended to even cores. However, these details should be verified by the user empirically.

7.6 Transparent Huge Pages

Transparent Huge Pages (THPs) are a Linux kernel feature for memory management to improve performance of the application by efficiently using processor's memory-mapping hardware. THP should reduce the overhead of the Translation Lookaside Buffer. User must login as a sudo user to enable or disable THP settings. It operates mainly in two modes:

- always: You can run the following command to set THP to ‘always’:

  ```
  echo always > /sys/kernel/mm/transparent_hugepage/enabled
  ```

  In this mode, the system kernel tries to assign huge pages to the processes running on the system.

- madvise: You can run the following command to set THP to ‘madvise’:

  ```
  echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
  ```

  In this mode, kernel only assigns huge pages to the individual processes memory areas.

You can use the following command to disable THP:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

It is recommended to use the following THP setting for better performance:

- CNN models - ‘always’
- NLP models - ‘madvise’

7.7 Memory Allocators

Based on the model, if there is a requirement for a lot of dynamic memory allocations, a memory allocator can be selected from the available allocators which would generate the most optimal performance out of the model. These memory allocators override the system provided dynamic memory allocation routines and use a custom implementation. They also provide the flexibility to override the dynamic memory management specific tunable parameters (for example, logical page size, per thread, or per-cpu cache sizes) and environment variables. The default configuration of these allocators would work well in practice. However, you should verify empirically by trying out what setting works best for a particular model after analyzing the dynamic memory requirements for that model.

Most commonly used allocator is JEMalloc.

7.7.1 JEMalloc

JEMalloc is a memory allocator that emphasizes fragmentation avoidance and scalable concurrency support. It has a powerful multi-core/multi-thread allocation capability. The more cores the CPU has, the more program threads, the faster JEMalloc allocates. JEMalloc classifies memory allocation granularity better, leading to less lock contention. It provides various tunable runtime options, such as enabling background threads for unused memory purging, allowing JEMalloc to utilize transparent huge pages for its internal metadata, and so on.

7.7.2 Usage

You can install the JEMalloc dynamic library and use **LD_PRELOAD** environment variable as follows:

```
Before using JEMalloc:
export LD_PRELOAD=/path/to/JEMallocLib/

Benchmarking command using JEMalloc:
LD_PRELOAD=/path/to/JEMallocLib/ < python benchmarking command>
```

To verify if JEMalloc memory allocator is in use, you can grep for tcmalloc/jemalloc in the output of `lsof` command:

```
lsof -p <pid_of_benchmarking_commad> | grep <jemalloc>
```

Chapter 8 License

ZenDNN is licensed under Apache License Version 2.0. Refer to the “LICENSE” file for the full license text and copyright notice.

This distribution includes third party software governed by separate license terms.

3-clause BSD License:

- Xbyak (*https://github.com/herumi/xbyak*)
- Googletest (*https://github.com/google/googletest*)
- Instrumentation and Tracing Technology API (*https://github.com/intel/ittapi*)

Apache License Version 2.0:

- oneDNN (*https://github.com/oneapi-src/oneDNN*)
- Xbyak_aarch64 (*https://github.com/fujitsu/xbyak_aarch64*)

Boost Software License, Version 1.0:

Boost C++ Libraries (*https://www.boost.org/*)

BSD/Apache/Software Licenses from PyTorch:

- License (*https://github.com/pytorch/pytorch/blob/master/LICENSE*)
- Notice (*https://github.com/pytorch/pytorch/blob/master/NOTICE*)

JEMalloc License:

https://github.com/jemalloc/jemalloc

This third-party software, even if included with the distribution of the Advanced Micro Devices software, may be governed by separate license terms, including without limitation, third-party license terms, and open-source software license terms. These separate license terms govern use of the third-party programs as set forth in the THIRD-PARTY-PROGRAMS file.

Intel ideep License:

https://github.com/intel/ideep/blob/master/LICENSE

Chapter 9 Technical Support

Please email *zendnnsupport@amd.com* for questions, issues, and feedback on ZenDNN.