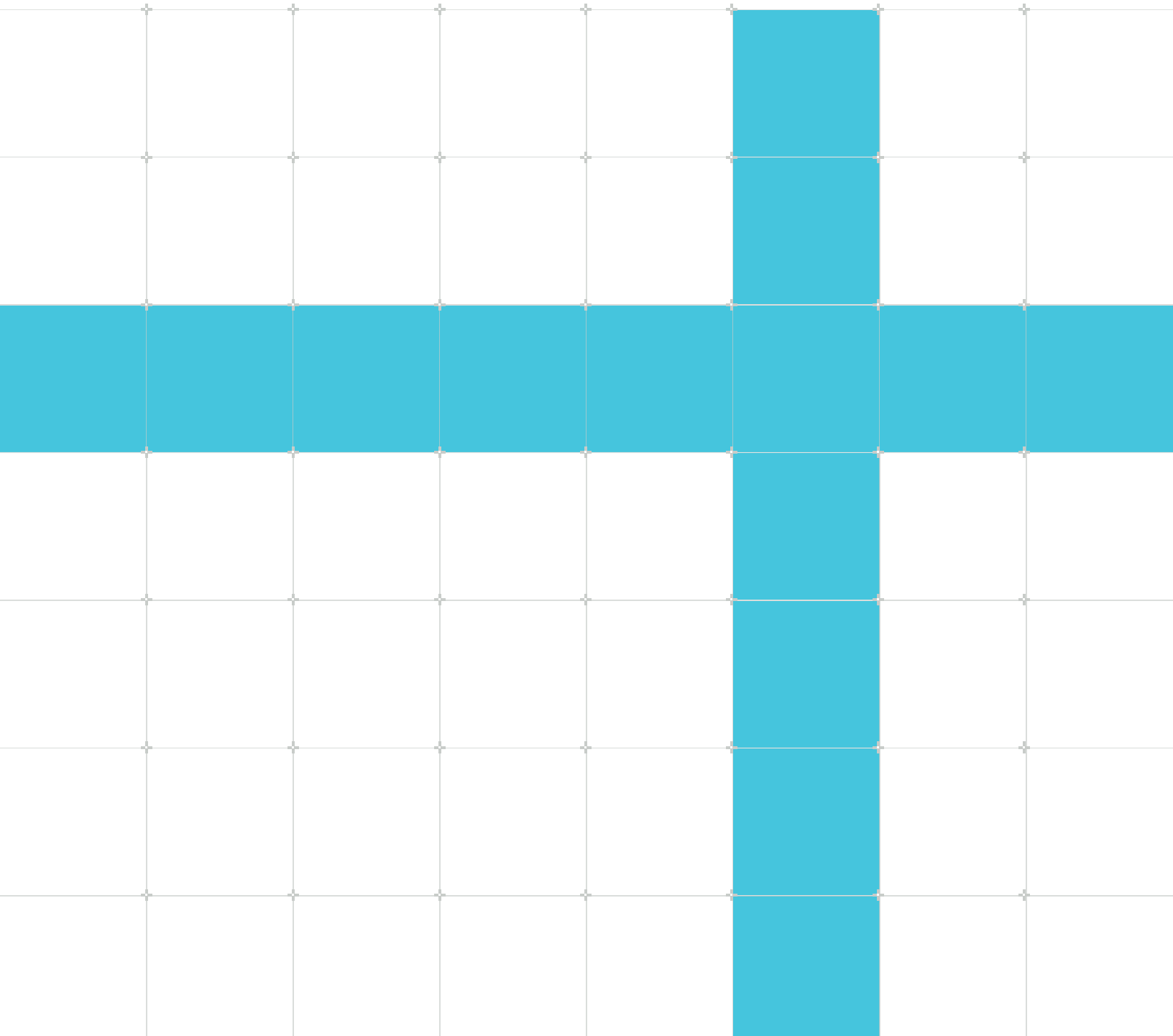
Arm® Neural Super Sampling

Version 1.0

Fine-tuning Guide

Issue 01
111141_0100_01_en



Arm® Neural Super Sampling Fine-tuning Guide

This document (111141_0100_01_en) was issued on 2025-10-17. There might be a later issue at https://developer.arm.com/documentation/111141

The product version is 1.0.

See also: Proprietary notice | Product and document information | Useful resources

Start reading

If you prefer, you can skip to the start of the content.

Intended audience

This document is for developers who want to train, fine-tune, and export neural graphics machine learning models, including upscaling techniques like Neual Super Sampling (NSS).

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Feedback

Arm welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com.

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Contents

1. Introduction

This guide describes how you can use the Arm® Neural Graphics Model Gym to fine-tune the Arm® Neural Super Sampling network to your own game content.

The Neural Graphics Model Gym is an open-source toolkit for developers who want to train, fine-tune, and export neural graphics machine learning models, including upscaling techniques like Neural Super Sampling (NSS).

1.1 Fine-tuning overview

Normally when we talk about machine learning models, we also talk about training them. The training process involves collecting a large amount of data and getting your model to learn to perform the intended functions from the training data.

With fine-tuning this process is shorter. We take a model trained on one dataset and then we make small adjustments to the model weights on a new, often smaller, or more specific dataset. This way the model adapts to the target task while the model retains the general knowledge that it has already learnt.

In other words, fine-tuning is training, but instead of starting from zero, we start near the finish line. Fine-tuning has the advantage of training the model faster when compared to training the model from zero.

This guide describes the process that you must perform to fine-tune the Neural Super Sampling (NSS) model for your game content. This includes:

- Best practices to capture training data for NSS.
- Information on how to convert captured NSS training data to Safetensors format.
- Information on how to install and use the Neural Graphics Model Gym.
- Best practices for setting training parameters.

1.2 Fine-tuning introduction

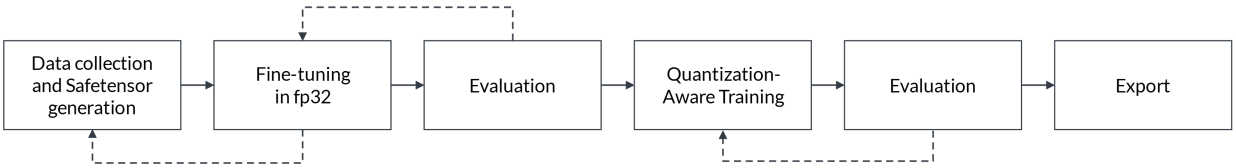
Arm provides checkpoint files for Neural Super Sampling (NSS) which are trained on a variety of different game content. The set of weights provides a good baseline for image quality and should work well when deployed in your own game.

To achieve the best image quality, you should fine-tune the NSS model using content captured from the game you are intending to integrate NSS in to.

Although NSS is trained on a variety of game content, NSS might not perform perfectly with your game. For example, your game might have a unique art style that was not covered in the training set. In this case, fine-tuning can help restore the expected image quality.

The following figure shows the workflow to fine-tune NSS and prepare it to use in your game.

Figure 1-1: Overview of NSS fine-tuning workflow



2. Neural Super Sampling dataset collection

As part of publishing Neural Super Sampling (NSS) and the Model Gym, we also released an NSS Data Specification. The specification outlines the data that you must capture from your game engine to train NSS. The specification also describes the format in which the data should be captured and how it should be laid out on disk.

2.1 Capture data

This section describes how to capture NSS training data in your game engine. In the future we will provide tools and examples which will describe how you can capture data.

Before you begin

Before capturing frames, check the requirements for using NSS in your game. Your game must have:

- Support for jittered rendering.
- Render motion vectors for most objects.
- Added or reused rendering options for very high resolution output.

We recommend integrating the runtime portion of NSS and verify that it runs correctly before proceeding with fine-tuning.

Locate the point in your engine's rendering pipeline before most post-processing effects are applied. For more information on the correct point in the rendering pipeline, see FidelityFX Super Resolution 2.3.3 (FSR2) | GPUOpen Manuals. Now follow these steps:

Procedure

1. Capture the very-high-resolution color, motion vector, and depth buffers.
2. To create the low-resolution input frames, pass the color, motion vector, and depth buffers into one or more shaders which decimate and jitter them.
3. To generate the high-quality ground-truth frames, downsample the color as recommended in the Dataset Specification.
4. Save these textures to disk in the layout as defined in the Dataset Specification. You do not need to continue the rendering pipeline after this step.
5. After you render the required number of frames, write a JSON file with required metadata. You can see an example in the Neural Graphics Model Gym.
6. Optionally, to capture frames of a pre-authored sequence, use **Replay** functionality in your game engine to simplify the capture process and make it repeatable.

2.2 What content to capture

When collecting training data for Neural Super Sampling (NSS), we recommend using a wide variety of scenes that reflect the types of content NSS must handle in your game.

Based on our experience developing NSS, we recommend capturing at least the following types of content:

Table 2-1: Different types of content

Content number	Content type	Goal	Comments
1	Static scenes	Ensure NSS can learn accumulation behaviour.	-
2	Real gameplay	Present NSS with realistic dynamic scenes that represent actual gameplay.	-
3	Static camera, moving light	Ensure NSS learns correct rectification and accumulation behaviour.	Also includes view-dependent lighting; can combine with moving cameras for harder content.
4	Dynamic camera motion	Ensure NSS learns correct rectification and accumulation behaviour.	-
5	Dynamic character motion	Ensure NSS learns correct rectification and accumulation behaviour.	-
6	Thin features	Present harder content to NSS. Ensures NSS learns to accumulate and not rectify.	Good to have both static and moving content.
7	High-contrast thin features	Present harder content to NSS. Ensures NSS learns to accumulate and not rectify.	Good to have both static and moving content.
8	Particle effects	Present harder content to NSS. Ensures NSS learns to rectify and not accumulate.	Moving content not tracked by motion vectors. Includes water effects like waterfalls.
9	High-frequency textures	Present harder content to NSS. Ensures NSS learns to accumulate and not rectify.	For example, grass, moss.
10	Reflections	Present harder content to NSS. Moving content in reflections lacks motion vectors; an extension of particle effects.	-
11	Foliage	Present harder content to NSS. Ensures NSS learns to accumulate and not rectify.	-
12	Transparent objects	Present harder content to NSS. Ensures correct accumulation and rectification behaviour.	-

Arm recommendations when preparing a training dataset:

Content type 1-5

60-70% of the total set of captured frames should be from gameplay that covers content types 1-5.

Content type 6-12

The remaining 30-40% of the captured frames should be from the harder content that is covered by types 6-12.

Although we recommend capturing specific types of content, it is most important to collect data that reflects actual gameplay. Ideally, the captured data includes a mix of content, for example, a moving camera, a moving character, particle effects, and complex textures.


Note

If you have indentified visual artifacts while using NSS in your game, you must capture the problematic scenes and include them as part of your training set.

2.3 Capture sequences

Neural Super Sampling (NSS) is a recurrent model that uses previous predictions and outputs to generate new outputs.

You must train NSS in a recurrent manner. The training input consists of a sequence of `t_train` consecutive frames. Typically, `t_train` is set to a value such as 16.

To generate the sequences of `t_train` frames, you must capture frame sequences from your game. The length of these captured sequences are labled as `t_captured`, where `t_captured` > `t_train`. Once collections of longer sequences have been captured, then the code within the model-gym handles the generation of batches of data of length `t_train` for the training process.

We recommend capturing many short sequences of frames rather than a few long ones. This approach simplifies dataset curation and increases training set diversity. We typically set `t_captured` to approximately 100.

2.4 How much data to capture

To train Neural Super Sampling (NSS), we use a dataset of more than 50,000 frames, divided into approximately 300 sequences. In general, larger datasets improve machine learning model performance.

If you do not need to train NSS from scratch, you likely do not need a large training dataset. Based on our experiments, fine-tuning NSS for your game requires about 5,000 frames to achieve good results.

3. Converting to Safetensors

Safetensors is a popular file format for storing tensor data. It is designed to be fast, secure, and memory efficient. The Model Gym requires training data in Safetensors format.

3.1 Install the Neural Graphics Model Gym

After you collected your training data, install the Neural Graphics Model Gym.

Before you begin

Ensure your system meets the requirements to use the Model Gym.

Procedure

1. Go to the Model Gym repository on GitHub.
2. Clone the repository to your system.

```
git clone https://github.com/arm/neural-graphics-model-gym.git
```

3. For installation, follow the guidance.

Results

At this point you should have:

- Captured sequences of frames.
- Cloned the Neural Graphics Model Gym.
- Installed the Neural Graphics Model Gym on your machine.

3.2 Converting captured data to Safetensors

Now you can convert your captured sequences to Safetensors.

To do this you can run the following command:

```
python -m scripts.safetensors_generator.safetensors_writer -src=path/to/exr/root/dir
```

The following table shows the optional flags in the script that you can set:

Table 3-1: Safetensor writer script flags

Flag	Description	Default
-dst	Path to root folder of destination.	./output/safetensors
-threads	Number of parallel threads to use.	1
-extension	File extension of the source data.	“exr”
-overwrite	Overwrite data in destination if it already exists.	False

Flag	Description	Default
-linear-truth	Whether the ground truth is already in linear color space. Assumes Karis tone mapped if not.	True
-logging_output_dir	Path to folder for logging output.	./output

Results

When the script finishes running, you will have a new folder with the Safetensors files in that location what is set by the `-dst` flag. Each Safetensors file will contain one complete sequence, based on the structure of the captured data folders.

3.3 Splitting data

After converting all the data to Safetensor format, it is up to you to split up the data into the following sets before proceeding:

Train

Data used for training the Neural Super Sampling (NSS) model.

Validation

Data used to periodically check how the training is progressing.

Test

Data used after the training to check the final quality of the model before usage.

A common practice is to split up the dataset into 70% training, 20% validation, and 10% testing. Although, it is crucial that this split is done fairly, so that each subset properly represents the overall data distribution.

Stratified random sampling is a more reliable approach than simple random shuffling. It preserves important characteristics across data splits, which leads to a more reliable training and evaluation outcomes.

When preparing data for training NSS, make sure that the distribution of the content types is similar across all data splits. This helps prevent issues such as having all the hard content in validation and testing sets, while leaving none for training.

3.4 Cropping

When capturing frames from a game, they are typically captured at full reolution. However, for training Neural Super Sampling (NSS), reduced-sized crops are used for training and validation. The preserved data for testing remains at full resolution.

To run cropping of your full resolution Safetensor files, you can run the following command:

```
python -m scripts.safetensors_generator.safetensors_writer -src="path/to/
safetensors/root/dir" -reader=cropper -extension=safetensors
```

Results

After running the cropping script, a new collection of cropped Safetensor files is created. Ensure that the following folders are present:

`train/`
: Contains the cropped safetensors that are used for training.

`val/`
: Contains the cropped safetensors that are used for validation.

`test/`
: Contains the cropped safetensors that are used for testing.

4. Running fine-tuning

The Neural Graphics Model Gym should now be installed, and the training data prepared. This includes Safetensor files cropped for training and validation, and full-resolution files preserved for testing.

You can now begin using the Model Gym to fine-tune the Neural Super Sampling (NSS) model.

4.1 Pretrained checkpoints

The Neural Super Sampling (NSS) release includes checkpoints that you can use with the Neural Graphics Model Gym. These checkpoints are the results of Arm NSS training. We provide an `fp32` checkpoint and a checkpoint that is the result of doing `int8` Quantization-Aware Training (QAT).

These checkpoints can be used to begin fine-tuning, which helps reduce training time and effort to capture a larger training dataset.

Download the `fp32` and the `int8` checkpoints from the Hugging Face Neural Super Sampling repository.

4.2 Calibrate the Model Gym configuration file

Neural Graphics Model Gym is configured using a JSON file. This configuration file contains all the necessary parameters to control the use of the Model Gym, such as paths to datasets and training hyperparameters.

Procedure

1. Generate the initial configuration JSON file:

   ```
   ng-model-gym init
   ```

   Two files are created:

   `schema_config.json`
   : Details all possible configuration options that can be changed.

   `config.json`
   : An example configuration file that you can use with the Model Gym.

2. Copy the `config.json` file content to use for running fine-tuning.

   ```
   cp config.json fine_tune_config.json
   ```

3. Open the `fine_tune_config.json` file in a text editor. Several options are already set.
4. Modify the following values:

`dataset.path.train`
: Set to the folder which contains your cropped training Safetensors file.

`dataset.path.validation`
: Set to the folder which contains your cropped validation Safetensors file.

`dataset.path.test`
: Set to the folder which contains your full resolution test Safetensor files.

`train.fp32.checkpoints.dir`
: Set to a folder where you wish to store the result from your training.

`train.qat.checkpoints.dir`
: Set to a folder where you wish to store the results from Quantization-Aware Training (QAT).

5. Set the following values to perform fine-tuning:

`train.finetune`
: Change this from `false` to `true`.

`train.pretrained_weights`
: Change this from null to the path of the fp32 checkpoint you downloaded from Hugging Face.


Note

Your personalized Model Gym configuration file can be named upon your preference when created, but for consistency reasons it will be referred as configuration file throughout this document.

Next steps

You are now ready to run fine-tuning.

4.3 Run fine-tuning

Now that your configuration file is set up, we will use the file as input to the Model Gym to perform fine-tuning. To run fine-tuning with the Model Gym, follow the steps in this topic.

Procedure

1. Use the Model Gym for fine-tuning. This command is telling the Model Gym to perform training:

```
ng-model-gym -c fine_tune_config.json train –finetune
```

2. The `--finetune` flag tells the Model Gym to load the checkpoints specified by `train, pretrained_weights`.
3. Fine-tuning starts with progress bars displayed. The progress bar shows the training process as it works through the training dataset.

4. The current loss value is printed to the progress bar. During the training, the loss value should decrease.
5. By default, validation is performed at the end of every epoch of training. An epoch is defined as one complete pass through the entire training dataset. When validating, image quality metrics are printed to the progress bar. During the training progress the image quality should improve.

The following list describes some of the image quality metrics.

Loss

The loss value quantifies how far the model predictions are from the true target values during the training. Lower the better.

PSNR

Peak Signal-to-Noise Ratio is a measure of image or signal quality that compares the maximum possible signal strength to the level of background noise. Higher the better.

tPSNR

Temporal PSNR is an extension of PSNR used for measuring video quality. It is the PSNR value calculated across the delta of a sequence of images. Higher the better.

recPSNR

Recurrent PSNR is PSNR that is calculated for the final image in a sequence of images. Higher the better.

SSIM

Structural Similarity Index measures the perceptual similarity between two images based on luminance, contrast and structural information. The higher the better.

Results

Once fine-tuning is complete, you will find a set of checkpoint files inside the location defined by `train.fp32.checkpoints.dir`.

4.4 Evaluation

After fine-tuning completes on the Neural Super Sampling (NSS) model, use the Model Gym to evaluate the quality on the NSS model. You can also compare the results against the original, pre-trained model from Hugging Face that was not fine-tuned on your dataset.

To evaluate the model, using Model Gym, run the following command:

```
ng-model-gym - c fine_tune_config.json evaluate –model-path=<path/to/your/ckpt.pt>
 --model-type=fp32
```

In this command:

`-c`

Like fine-tuning, we pass the configuration file in with `-c` as the first parameter.

`evaluate`
: Instructs the Model Gym to run evaluation.

`--model-path`
: Provides the path to the checkpoint saved after fine-tuning.

`--model-type`
: Defines the model type as `fp32`.

A progress bar appears followed by image quality metrics. Evaluation runs on the entire test dataset defined in the configuration file. Once complete, the results are displayed and saved to the output folder set in the configuration file.

To compare with the baseline model, repeat the procedure, using the checkpoint downloaded from Hugging Face by changing the `-model-path`. The fine-tuned model should show improved performance on the test dataset than the Hugging Face model. If the fine-tuned model does not perform better than the Hugging Face model, consider collecting additional training data or adjusting the training parameters before fine-tuning again. For more information about adjusting the parameters, see Best practices and common issues.

4.5 Quantization-Aware Training

For maximum precision and stability, we recommend fine-tuning in `fp32`. When deploying the model on devices with neural accelerators, you must quantize the model weights to `int8` format.

About this task

To quantize the model weights, you can use the Model Gym to perform Quantization-Aware Training (QAT). QAT simulates quantization during training while allowing the model weight to adapt to the reduced precision. This helps preserve the accuracy and quality of the original `fp32` floating-point-model.

To perform QAT with the Model Gym, follow these steps.

Procedure

1. Update the `train.pretrained_weights` key in the configuration file.
2. Replace the downloaded `fp32` checkpoint values with the path to the best `fp32` checkpoint obtained after fine-tuning.
3. Run the following command with the Model Gym:

   ```
   ng-model-gym -c fine_tune_config.json qat -finetune
   ```

   This command is similar to the one used for fine-tuning. The main difference is the use of `qat` instead of `train`.

   A progress bar appears to track the training process.
4. After QAT completes, a set of checkpoint files are saved in the defined directory by the `train.qat.checkpoints.dir` key.

5. Use the following command to run evaluation on the QAT model:

```
ng-model-gym - c fine_tune_config.json evaluate –model-path=<path/to/your/
ckpt.pt> --model-type=qat_int8
```

Results

The QAT model should produce similar evaluation results to the fined-tuned `fp32` model.

Next steps

If the results show lower accuracy, collect additional training data. Collect the training data before repeating the training or adjusting the QAT parameters as described in Best practices and common issues.

4.6 Export

After the Quantization-Aware Training (QAT) completes and the Neural Super Sampling (NSS) model meets the quality expectations, export the model to `vgf`. This format allows you to use the NSS model with the ML Extensions for Vulkan and to use in your game through Unreal Engine plugin.

To export the model, run the following command:

```
ng-model-gym -c fine_tune_config.json export –model-path=<path/to/your/ckpt.pt> --
export-type=qat_int8
```

In this command:

`-c`
: The first parameter that helps to pass the configuration file on.

`export`
: Commands the Model Gym to perform exporting to `vgf`.

`--model-path`
: Gives the path to the checkpoint file to export.

`--export-type=qat_int8`
: Specifies that the exported checkpoint is a result of the QAT.

After the export completes, a `.vgf` file is saved in the location defined in the configuration file.

5. Best practices and common issues

You might encounter some issues when fine-tuning your custom data. This section shows you the best practices and common issues you might face and how to resolve them.

5.1 Use TensorBoard

The Model Gym integrates TensorBoard into the training and evaluation flows. When you start training or fine-tuning you should see the following in the console:

```
[INFO] TensorBoard access command: $ tensorboard --logdir /path/to/neural-graphics-model-gym/tensorboard-logs
```

You can follow this command to run TensorBoard and view graphs of the training statistics, for example, training and validation loss as they change over time. This is a good way to see trends in your training process and help you identify any problems.

5.2 Set values for the training Hyperparameters

Training neural networks involves tuning many parameters that affect the outcome, for example, the batch size, optimizer, and learning rate. This process can be overwhelming for beginners.

To help with training and finetuning of Neural Super Sampling (NSS) we have supplied values for all hyperparameters in the default generated configuration file.

If the dataset captured are large enough, and of good quality, then these default values should work well and allow you to train and finetune NSS.

5.3 Training loss is not decreasing

During fine-tuning, the training loss should steadily decrease over time.

Cause

If the training loss does not decrease, several factors might be responsible. To help solve this problem, try the following methods.

Solution 1

1. Confirm that the data is captured correctly.
2. In the configuration, set `dataset.health_check` to true.

This enables the Model Gym to perform basic checks for common issues in data loaded from Safetensor files.

Solution 2

1. Decrease the learning rate. If you see that the loss value is increasing or bouncing around, then this might indicate that your learning rate is set too high.
2. Try slowly reducing the learning rate by a factor of 2 and performing fine-tuning again.
3. Do this by adjusting the value in the configuration file at either option:
   - `train.fp32.optimizer.learning_rate`
   - `train.qat.optimizer.learning_rate`
4. We automatically decrease the learning rate over time according to a schedule. In the configuration file, the following parameters define what the minimum learning rate can be:
   - `train.fp32.lr_scheduler.min_lr`
   - `train.qat.lr_scheduler.min_lr`


Note

- You might need to reduce the minimum learning rate if you are adjusting the base learning rate value.
- If you set the learning rate too low, then you might run into the opposite problem. Your loss value might barely decrease or remain constant.

5.4 Training loss is decreasing but validation loss is not

You might find that your training loss decreases as expected, but the loss on the validation set does not decrease.

Cause

This is generally a sign that overfitting to the training data is happening. To resolve overfitting, try the following methods.

Solution 1

Ensure that the training and validation data distributions match. When constructing your training and validation sets, they might not have been well mixed. This could result in fine-tuning on one type of data but validating on something completely different that the model has not been able to generalise to.

Solution 2

1. Increase the size and variety of your training data set.
2. Collect more training samples and remember to create new training, validation, and test splits.

Solution 3

Reduce the learning rate. If your validation loss has reached a plateau and is not decreasing anymore, reduce the learning rate to help convergence.

Solution 4

Early stopping. When validation loss stops improving, stop the training early.

5.5 Out of memory while training

Currently, the training process for Neural Super Sampling (NSS) requires a large amount of GPU memory to complete. This is due to the use of Learned Perceptual Image Patch Similarity (LPIPS) metric in the loss function and the recurrent nature of training.

The recommended batch size (16) and number of recurrent samples (16) can result in a relatively high memory usage of around 40GB of GPU memory. To help reduce the memory usage, you can reduce either the batch size or the number of recurrent samples. You can also reduce both at the same time.

Reducing the parameter values may affect the quality of the trained NSS model. You must carry out evaluation to check the effect on the fine-tuned model.

Proprietary Notice

This document is protected by copyright and other related rights and the use or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm Limited ("Arm"). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether the subject matter of this document infringes any third party patents.

The content of this document is informational only. Any solutions presented herein are subject to changing conditions, information, scope, and data. This document was produced using reasonable efforts based on information available as of the date of issue of this document. The scope of information in this document may exceed that which Arm is required to provide, and such additional information is merely intended to further assist the recipient and does not represent Arm's view of the scope of its obligations. You acknowledge and agree that you possess the necessary expertise in system security and functional safety and that you shall be solely responsible for compliance with all legal, regulatory, safety and security related requirements concerning your products, notwithstanding any information or support that may be provided by Arm herein. In addition, you are responsible for any applications which are used in conjunction with any Arm technology described in this document, and to minimize risks, adequate design and operating safeguards should be provided for by you.

This document may include technical inaccuracies or typographical errors. THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Reference by Arm to any third party's products or services within this document is not an express or implied approval or endorsement of the use thereof.

This document consists solely of commercial items. You shall be responsible for ensuring that any permitted use, duplication, or disclosure of this document complies fully with any relevant

export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of this document shall prevail.

The validity, construction and performance of this notice shall be governed by English Law.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

PRE-1121-V1.0

Product and document information

Read the information in these sections to understand the release status of the product and documentation, and the conventions used in Arm documents.

Product status

All products and services provided by Arm require deliverables to be prepared and made available at different levels of completeness. The information in this document indicates the appropriate level of completeness for the associated deliverables.

Product completeness status

The information in this document is for a product under development (dev product).

Revision history

These sections can help you understand how the document has changed over time.

Document release information

The Document history table gives the issue number and the released date for each released issue of this document.

Document history

Issue	Date	Confidentiality	Change
0100-01	17 October 2025	Non-Confidential	First release for version 1.0

Change history

The Change history tables describe the technical changes between released issues of this document in reverse order. Issue numbers match the revision history in Document release information on page 23.

Table 2: Issue 01

Change	Location
First release for version 1.0	-

Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: developer.arm.com/glossary.

Typographic conventions

Arm documentation uses typographical conventions to convey specific meaning.

Convention	Use
italic	Citations.
bold	Interface elements, such as menu names. Terms in descriptive lists, where appropriate.
`monospace`	Text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace underline`	A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`<and>`	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Terms that have specific technical meanings as defined in the *Arm® Glossary*. For example, **IMPLEMENTATION DEFINED**, **IMPLEMENTATION SPECIFIC**, **UNKNOWN**, and **UNPREDICTABLE**.

Caution

We recommend the following. If you do not follow these recommendations your system might not work.

Warning

Your system requires the following. If you do not follow these requirements your system will not work.

You are at risk of causing permanent damage to your system or your equipment, or harming yourself.

This information is important and needs your attention.

A useful tip that might make it easier, better or faster to perform a task.

A reminder of something important that relates to the information you are reading.

Useful resources

This document contains information that is specific to this product. See the following resources for other useful information.

Arm documents are available on developer.arm.com/documentation.

Confidential documents are only available to licensees, when logged in. Each document link in the tables below provides direct access to the online version of the document.

Arm product resources	Document ID	Confidentiality
Arm® Get started with neural graphics using ML Extensions for Vulkan® Learning Path	–	Non-Confidential
Arm® Hugging Face Neural Super Sampling checkpoint files	–	Non-Confidential
Arm® Neural Graphics Model Gym	–	Non-Confidential
Arm® Neural Graphics Model Gym examples	–	Non-Confidential
Arm® Neural Super Sampling Dataset Specification	–	Non-Confidential

Non-Arm resources	Document ID	Organization
Safetensors	–	Safetensors
TensorBoard	–	TensorFlow