WEKA®

NeuralMesh™ by WEKA® for Checkpointing

Faster Recovery, Better GPU Utilization

Table of Contents

Abstract

How much does storage latency really impact the performance of checkpointing in AI training pipelines? To find out, we benchmarked NeuralMesh against a high-performance NFS-based storage system using NVIDIA's NeMo toolkit across three model types: FastPitch (TTS), ASR (speech-to-text), and BioMegatron (NLP). The tests were run in a controlled AWS environment with consistent GPU hardware and network configurations to isolate storage performance under real-world checkpointing conditions.

Results show that NeuralMesh consistently delivered 40–50% lower checkpointing latency across all workloads, enabling more frequent checkpoints without stalling training jobs. These findings highlight the critical role of low-latency, mixed-IO-capable infrastructure in supporting reproducible, scalable, and cost-efficient AI model development.

This is exactly where NeuralMesh excels—delivering the sustained throughput needed for aggressive checkpointing and the flexibility to handle mixed workloads without tuning, in any deployment environment

Introduction

Modern AI pipelines don't just demand scale—they demand resilience. As training runs grow longer, more complex, and more expensive, the ability to checkpoint frequently and recover quickly has become essential to delivering models on time, within budget, and without compromise. Whether you're training on a small cluster or scaling across thousands of GPUs, one truth holds: If your checkpointing fails, your pipeline stalls.

What was once a best practice in academic research is now a critical requirement in enterprise AI. Training workflows must hit strict service level objectives (SLOs) for runtime, fault tolerance, and reproducibility. That means infrastructure can no longer afford to treat checkpointing as a secondary concern—it has to be fast, reliable, and seamless under load.

Yet in many environments, the storage layer becomes the bottleneck. Legacy systems struggle to sustain low-latency writes across mixed I/O profiles, introducing stalls that idle GPUs and slow experimentation. Worse, those delays often go unnoticed until training grinds to a halt or a failure forces a full restart from scratch.

This paper explores how NeuralMesh eliminates that risk. We'll examine real-world benchmark data comparing checkpointing performance on NeuralMesh vs. traditional NFS-based infrastructure using NVIDIA NeMo and a range of model types. The results show that NeuralMesh delivers dramatically lower latency, enabling more aggressive checkpointing, faster recovery, and higher GPU utilization—so teams can iterate faster and recover smarter.

Checkpointing: The Tradeoff between Resilience and Speed

The more frequently you checkpoint, the more you gain in resilience and reproducibility. But each checkpoint is a write operation that contends with the rest of your pipeline's I/O: ingest, preprocessing, validation, fine-tuning, archiving. And when infrastructure can't keep up, those checkpoints introduce stalls that ripple across your training runtime.

Here's the tradeoff:

- **Checkpoint too infrequently,** and every failure forces you to re-run hours—or days—of training
- **Checkpoint too frequently,** and you risk overwhelming the storage system, stalling training jobs, and wasting expensive GPU cycles

This is especially true in production-scale environments where training pipelines are distributed, multi-modal, and non-deterministic. Cloud-hosted GPUs can be reallocated mid-run. Code updates

and data prep errors introduce non-linear faults. And with models growing in size and complexity, recovery from scratch isn't just a setback—it's a budgetary failure.

The goal isn't to checkpoint more or less. It's to checkpoint intelligently—as often as your failure domains demand, without being bottlenecked by your storage stack.

The ability to make that tradeoff disappear—checkpointing as often as needed, without stalling training—is where NeuralMesh shines. Its architecture absorbs high-frequency checkpoint writes while simultaneously servicing reads, metadata ops, and other I/O streams, so the rest of your pipeline stays fully utilized.

The Real Cost of a Stalled Checkpoint

Even a few seconds of added latency per checkpoint can snowball into hours of wasted runtime—especially in multi-GPU, distributed environments. Here's what's at stake:

- Missed SLAs due to unpredictable job completion times
- Lost productivity as engineers wait to rerun failed training
- Compromised reproducibility when checkpoints are skipped or corrupted
- Pipeline instability as mixed I/O workloads collide under load

Checkpointing shouldn't be a bottleneck. **With NeuralMesh, it isn't.**

Why Legacy Storage Stalls Your AI Pipeline

In theory, checkpointing should be lightweight. In practice, it often stalls progress. Checkpointing doesn't just test your training pipeline—it tests your infrastructure. And for many AI teams, that's where the real friction starts.

Each checkpoint introduces a burst of latency-sensitive writes. When your storage system can't absorb that I/O load fast enough, everything upstream—data ingest, training, validation—backs up. That's because traditional enterprise storage wasn't built for AI's mixed I/O demands. AI pipelines are not sequential—they're parallel, bursty, and unpredictable. A single system might be simultaneously:

- Reading training data
- Writing checkpoints
- Validating outputs
- Archiving artifacts

...and doing it across multiple concurrent jobs.

Legacy NFS-based systems struggle to keep up. Their metadata operations bottleneck under load. Their write paths weren't designed for microsecond-level responsiveness. Their performance tuning is brittle and workload-specific.

When this happens:

- Checkpoints lag behind model state
- GPUs wait on I/O

What's needed isn't just faster throughput. It's infrastructure that can handle any I/O pattern—reads, writes, metadata, large and small files—with consistent low latency across every phase of the pipeline.

That's where NeuralMesh is fundamentally different: it's built to sustain both massive throughput and ultra-low latency under any I/O profile, ensuring that checkpointing never competes with the rest of your pipeline for performance.

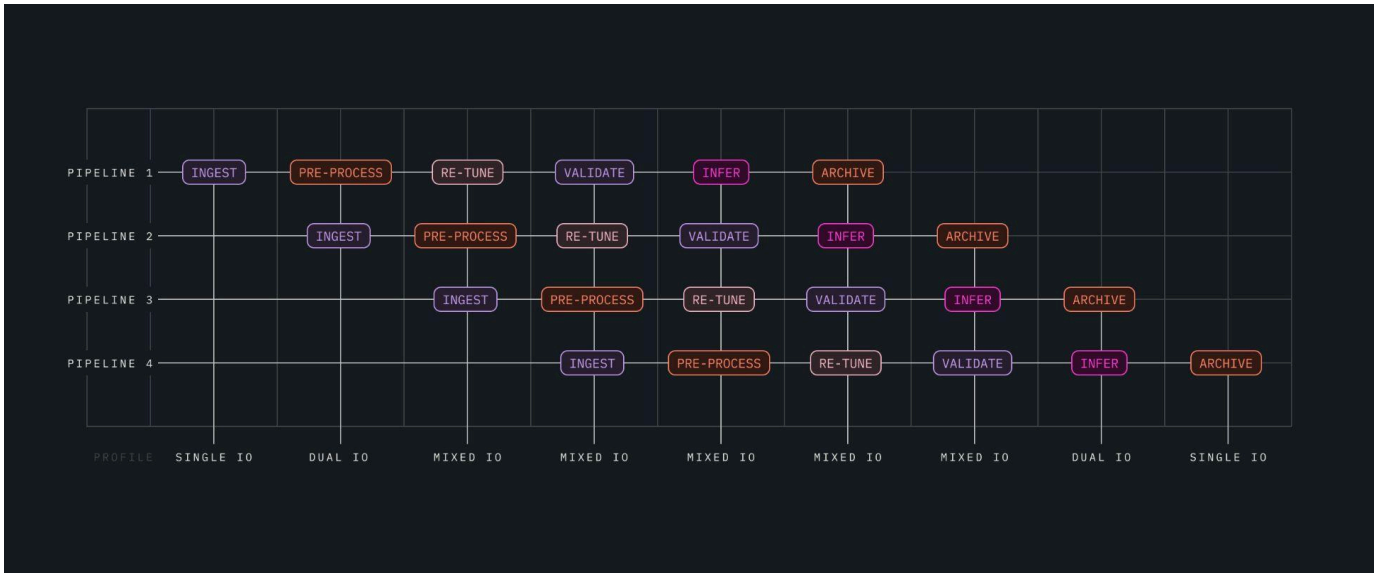

Fig. 1: Pipeline diagram

How NeuralMesh Eliminates the Checkpointing Bottleneck

NeuralMesh is specifically designed for diverse concurrent workloads with a high-performance file system that services AI/ML workloads with open standards. NeuralMesh's ability to run diverse concurrent workloads distinguishes it from other legacy storage by providing industry leading metadata performance, scale out read and write performance on large and small file operations, as well as operational efficiency when storing billions of small files.

Many elements of a workflow require each step to complete before you can reach the end of the job. With checkpointing specifically, training on a model cannot complete until the checkpoint has finished writing. If a checkpoint requires performance that exceeds a legacy storage system's capability, job runtime can slow significantly and the user starts asking the question again: "why isn't it running as fast as I need?"

NeuralMesh's services provide concurrent operations for all data intensive workloads that need accelerated data. With the ease of creating data experiments on as small an environment as your laptop, imagine the concurrency of tens or hundreds or thousands of experiments on enterprise level infrastructure. NeuralMesh can handle this I/O blender with ease.

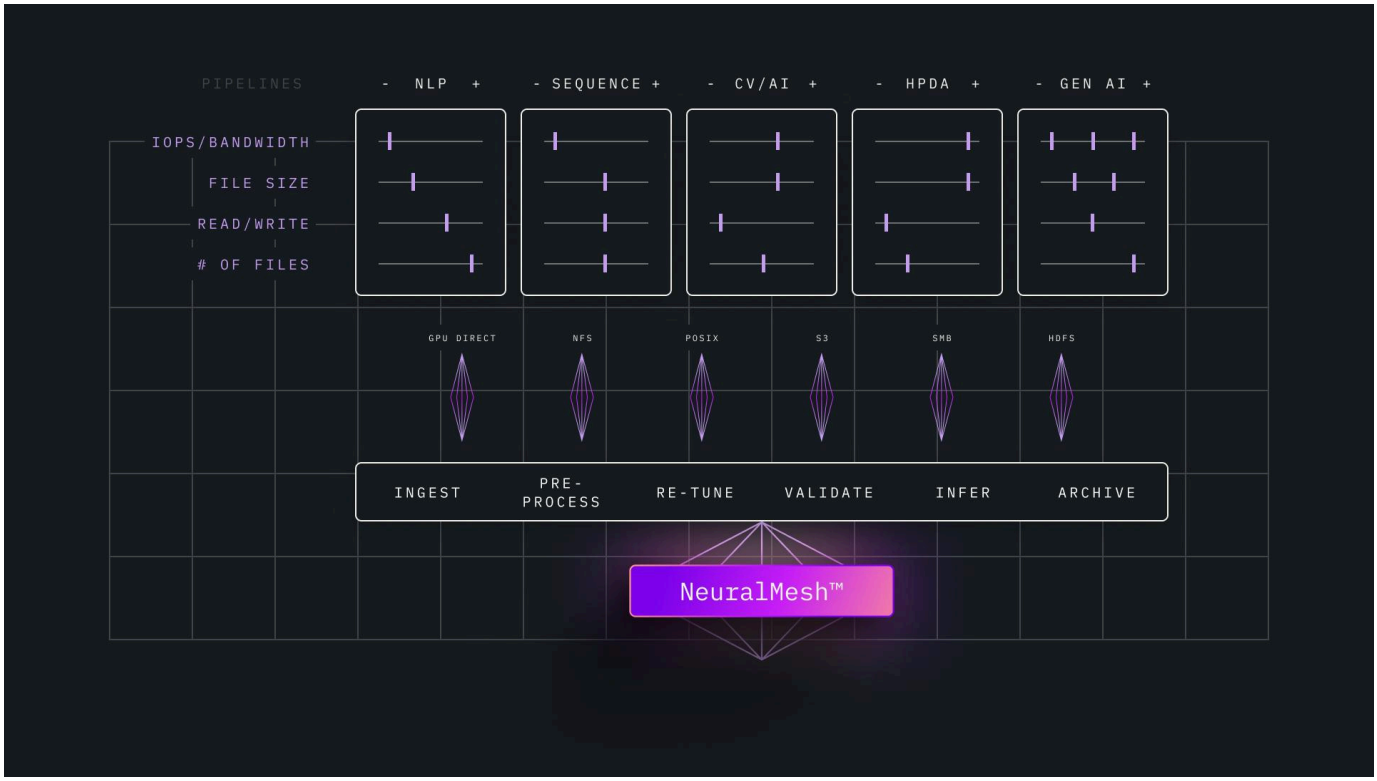

Fig. 2: Workload diagram

Checkpoint Performance Testing

To evaluate checkpointing performance, we used NVIDIA's NeMo framework —a widely adopted generative AI toolkit that supports end-to-end model development across multiple modalities. NeMo is representative of the kind of production-grade toolchains AI teams rely on to train, fine-tune, monitor, and checkpoint large models.

NVIDIA designed NeMo to streamline the entire AI pipeline, offering tools for training, telemetry, observability (via TensorBoard), and robust checkpointing. It is optimized for GPU-based workloads

and tightly aligned with the NVIDIA NGC ecosystem, making it ideal for benchmarking infrastructure performance under real-world conditions.

Checkpointing in NeMo is designed to support incremental training and fast recovery. When a failure occurs—whether due to a data error, code issue, or infrastructure fault—NeMo allows training to resume from the most recent checkpoint. This minimizes GPU idle time, accelerates iteration, and reduces the risk of lost work.

However, NeMo is only as fast as the infrastructure beneath it. Its checkpointing capabilities depend heavily on the performance of the underlying storage layer. Slow writes or high latency can stall training jobs, negate productivity gains, and inflate infrastructure costs.

Process and Models Used

For this test, we built a NeMo environment and trained three different models to assess checkpointing performance across varying levels of complexity:

- **FastPitch**: A parallel text-to-speech model (small)
- **ASR**: A speech-to-text model (medium)
- **BioMegatron**: A large MegatronBERT-based NLP model (large)

This range reflects typical workloads in production AI environments and allows us to evaluate how checkpointing scales with model size and structure.

All models were trained using best practices for checkpointing within NeMo, and documentation for the full training setup—including model parameters and logs—is provided in the appendix.

NeuralMesh Integration

Integrating NeuralMesh with NeMo is straightforward. NeuralMesh can be deployed on-premises, in any major hyperscale cloud, or in hybrid environments. Compute clients can access the platform via standard protocols including POSIX, NFS, SMB, and S3, as well as NVIDIA GPUDirect Storage (GDS). For containerized environments, NeuralMesh also provides a CSI driver to mount volumes directly into Kubernetes pods.

From the user's perspective, NeuralMesh behaves like a high-performance NVMe-tier filesystem—with the scalability and flexibility to handle petabyte- to exabyte-scale datasets. It delivers the consistency and low latency of local storage, but with the durability and operational maturity required for enterprise AI workflows.

Once NeMo is configured to use a NeuralMesh POSIX client, the entire pipeline—from data prep to training, tuning, and deployment—runs seamlessly. Checkpoint creation and recovery behave exactly as expected, with no additional tuning or workflow changes required.

Test Environment

To benchmark checkpointing performance, we built a small NeuralMesh cluster in AWS using six i3en.2xlarge EC2 instances. The training jobs accessed NeuralMesh via the POSIX client, enabling high-performance file access from the GPU.

For comparison, we also deployed a high-performance NFS server on a dedicated c5n.18xlarge instance.

Training was conducted on a G4dn.8xlarge instance equipped with a single NVIDIA T4 GPU (16GB vRAM). All components were selected to ensure that network throughput would not be a bottleneck during testing.

Latency was measured using NeuralMesh's native stats function, which tracks round-trip times between client and cluster. For the NFS server, we captured latency data using Tshark (a terminal-based version of Wireshark), measuring checkpoint completion time via line captures.

Methods: Measuring Write Latency

To evaluate write performance during checkpointing, we followed best practices for each training run and instrumented the system to measure latency at each checkpoint commit.

NeuralMesh latency was measured using built-in NeuralMesh stats tools to capture round-trip write times from the client to the cluster. For the NFS server, we used Tshark to perform packet-level line captures and determine checkpoint completion times.

We evaluated three models—representing small, medium, and large AI workloads—under the same test environment.

Continue to the next page...

FastPitch Training (Small Model)

In the FastPitch fine-tuning run, checkpoints were triggered frequently—roughly every 10 seconds. As shown in Figure 3, NeuralMesh handled these writes with an average latency of approximately **900 microseconds**, while the NFS server came in nearly twice as slow at **~1850 microseconds**.

This test demonstrates NeuralMesh's ability to sustain low-latency writes even under high-frequency checkpointing conditions. Sub-millisecond performance is a direct outcome of NeuralMesh's ability to handle high-frequency, bursty writes without disrupting other workloads—critical for efficient checkpointing in multi-job environments.

Documentation of how to run a FastPitch training with NeMo including the specific model parameters used in this example and run logs is listed in the appendix of this paper.

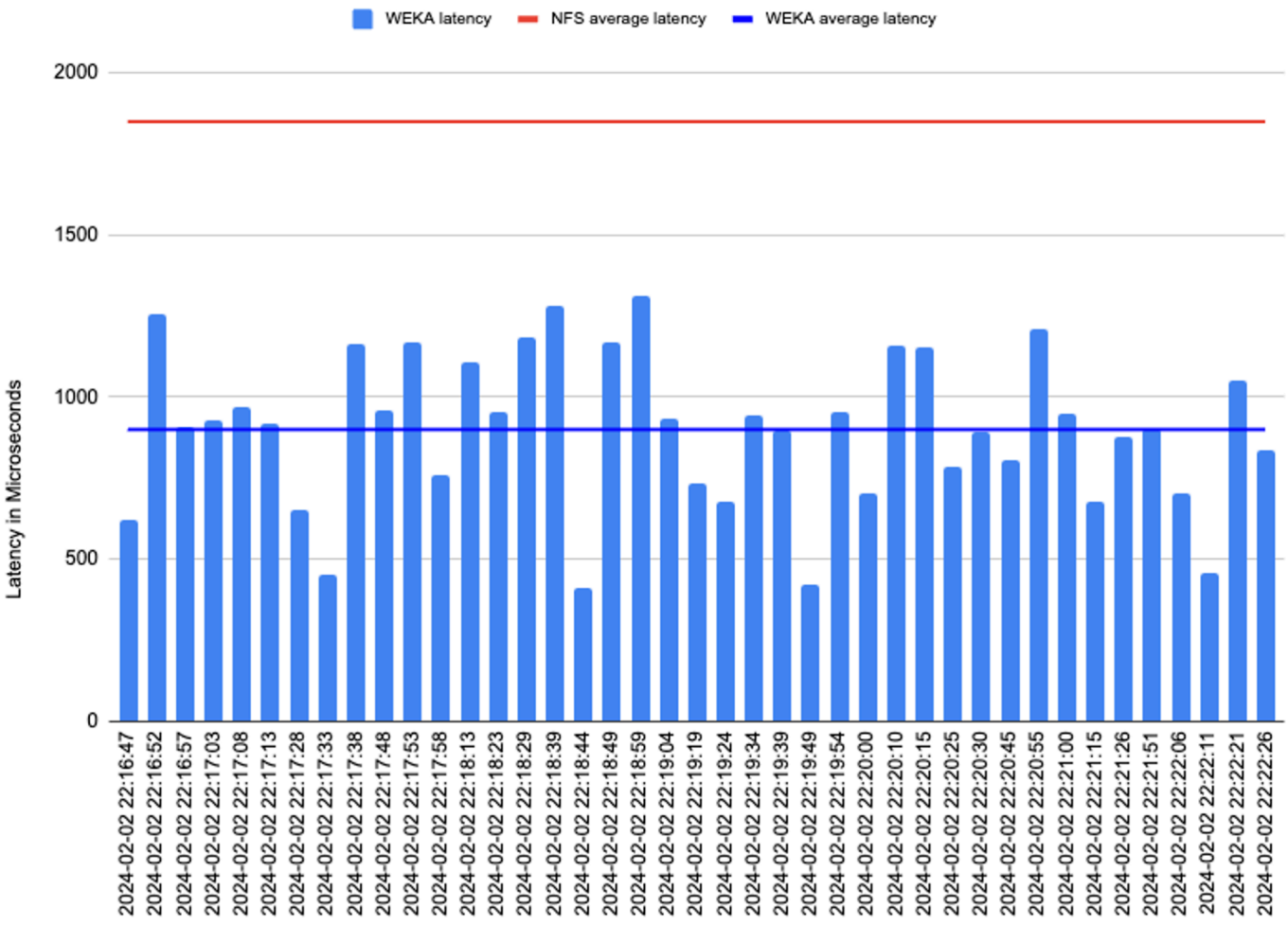

Fig. 3: Latency, FastPitch training

ASR Training (Medium Model)

The ASR model exhibited less predictable checkpointing intervals due to the model's structure and training dynamics. Some checkpoints were spaced over 30 seconds apart, while others occurred within 5 seconds.

Even under these variable conditions, NeuralMesh consistently delivered checkpoint write latencies of **~900 microseconds**, compared to **~1825 microseconds** on NFS (Figure 4).

That consistency under variable intervals shows NeuralMesh's mixed-workload strength. Whether checkpoints arrive every 5 seconds or 30, NeuralMesh sustains throughput so other pipeline processes remain unaffected.

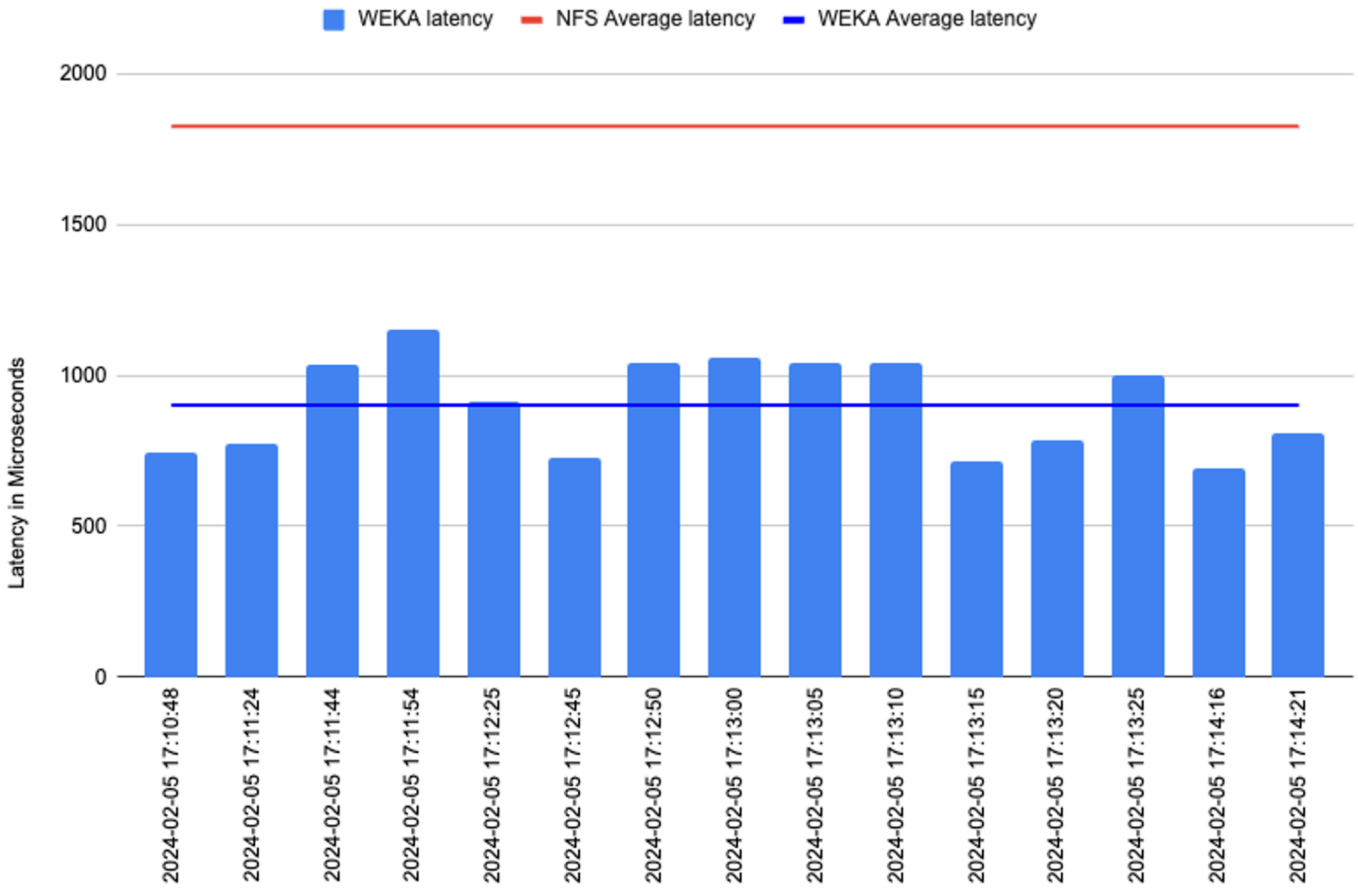

Fig. 4: Latency, ASR training

BioMegatron Training (Large Model)

BioMegatron presented the most unpredictable checkpoint behavior, with intervals ranging from 4 to over 45 seconds. Despite the increased workload and larger model size, NeuralMesh maintained a write latency of approximately **1090 microseconds**, while the NFS server remained at **~1850 microseconds** (Figure 5).

When training large-scale models on distributed infrastructure, latency becomes a critical gating factor. Delays in checkpoint commits can cascade across GPUs, stalling progress and reducing overall system efficiency.

But even at large model scale, NeuralMesh absorbs massive checkpoint writes while servicing concurrent I/O—something traditional NFS systems can't match without performance collapse. This is where NeuralMesh's architecture pays off. By sustaining both high throughput and low latency across mixed workloads, it ensures that every GPU remains productive—whether you're running a single model or hundreds of experiments in parallel.

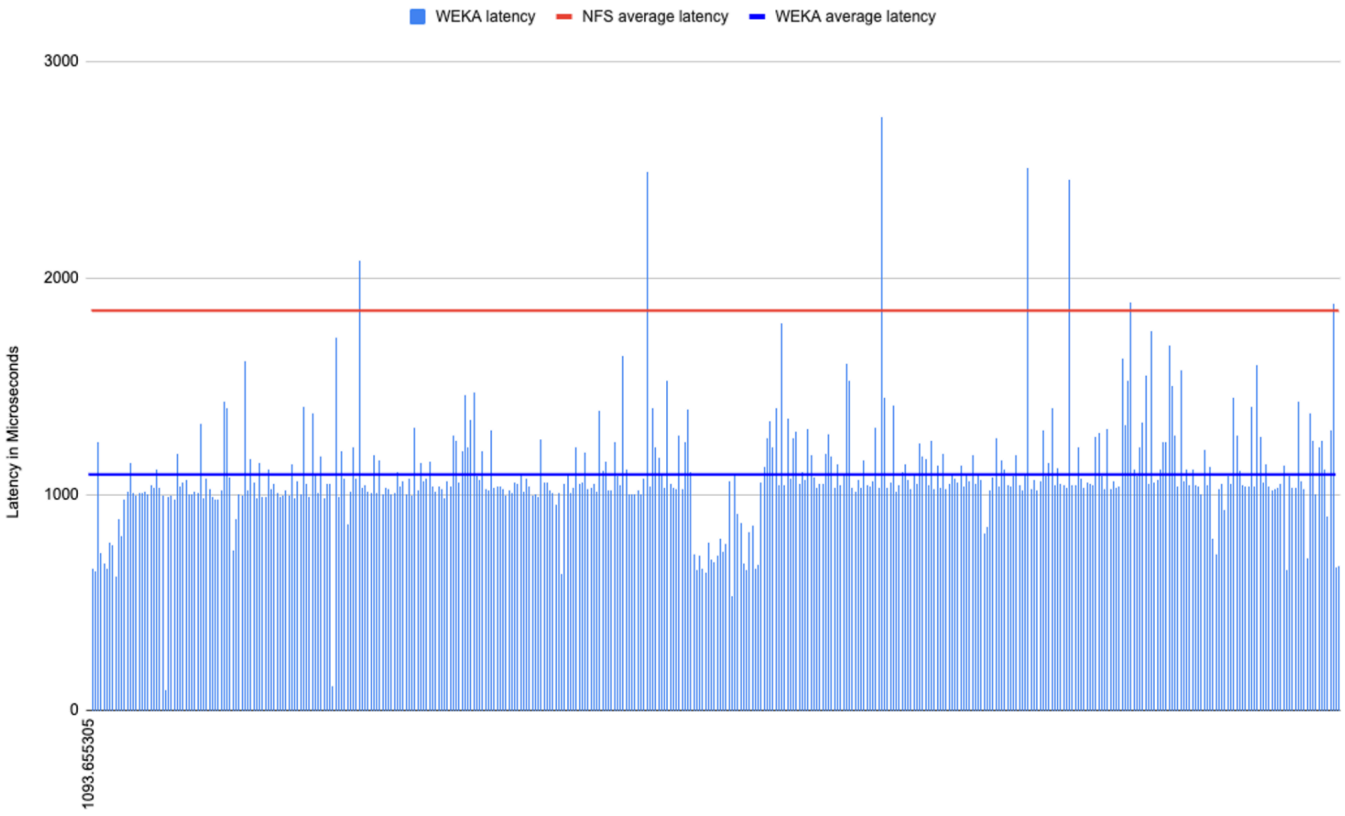

Fig. 5: Latency, BioMegatron training

Results

Across all three model tests, NeuralMesh consistently outperformed the NFS server on checkpoint write latency—**by as much as 50%**. While the absolute differences may appear small in single-GPU scenarios, they scale dramatically in multi-GPU and distributed training environments.

For example, with high-end GPUs like the NVIDIA GB200, which operate hundreds of times faster than the T4 used in our test environment, checkpoint frequency and throughput demands rise significantly. In distributed training, the latency of a single checkpoint operation can stall every GPU in the cluster until the write completes.

This makes low, consistent latency a key enabler of pipeline efficiency. Faster checkpoints mean:

- Less idle GPU time
- Faster failure recovery
- Greater flexibility in how often you checkpoint
- Lower risk of data loss due to unexpected interruptions

Checkpointing isn't just a storage benchmark—it's a productivity multiplier. And in every test, NeuralMesh demonstrated the ability to sustain aggressive checkpointing workloads while keeping latency reliably low.

Conclusion

AI infrastructure built for legacy enterprise workloads simply wasn't designed to meet the demands of modern training pipelines. As a result, teams are forced to make tradeoffs—limiting how often they checkpoint, accepting stalled training jobs, or building workarounds to accommodate unpredictable performance. That model doesn't scale.

Checkpointing is a critical function for ensuring training resilience, budget predictability, and throughput at scale. If your storage layer can't keep up, it becomes a liability—one that directly impacts time-to-market and total cost of AI operations.

NeuralMesh removes that bottleneck and delivers on two fronts:

1. **Throughput for aggressive checkpointing**: Write checkpoints as often as you need without stalling training jobs or idling GPUs.
2. **Mixed-workload performance anywhere**: Maintain predictable, low-latency performance across any I/O pattern—in the cloud, on-prem, or hybrid—without tuning or complex setup.

By delivering consistently low-latency writes—even under mixed I/O loads and unpredictable checkpointing patterns—NeuralMesh enables teams to checkpoint as often as needed without stalling progress. Compared to high-performance NFS, NeuralMesh cut write latency nearly in half

in our tests, translating directly into better GPU utilization, faster recovery, and more productive iteration cycles.

In AI, time is money—and infrastructure that accelerates checkpointing is a competitive advantage.

NeuralMesh

A Better Solution for Checkpointing under Pressure

The ability to rapidly recover and prevent GPUs from re-doing hours or days of work can't be ignored: NeuralMesh's latency advantage translates to real world savings. In contrast to NFS-based solutions for model training, NeuralMesh has proven itself to have significantly lower latency—as well as being able to handle any I/O profile of workload thrown at it.

Here's how NeuralMesh stacks up against traditional NFS-based storage when it comes to handling the demands of AI training pipelines:

Capability	Legacy NFS Storage	NeuralMesh
Checkpoint frequency	Must be tuned to avoid stalls; frequent checkpointing can overwhelm the system	No restrictions—checkpoint as often as needed without performance degradation
Mixed I/O workload performance	High variability; metadata and concurrent jobs often introduce latency and contention	Consistent low latency for reads, writes, and metadata—even across diverse I/O profiles
System tuning requirements	Requires workload-specific tuning and often multiple mount points per job type	Single-mount architecture with DevOps-grade observability and no manual tuning required
Scalability and deployment flexibility	Typically limited to static environments or complex multi-instance setups	Runs anywhere—cloud, on-prem, or hybrid—with elastic scaling and Kubernetes-native support

Checkpointing Built for the AI Stack

While this paper focuses on checkpointing, NeuralMesh is designed to meet the full lifecycle demands of AI development at scale:

- **Zero tuning required**: Run any workload without low-level configuration. Let researchers focus on models, not mounts.
- **Hybrid-ready**: Seamlessly operate across on-premises and cloud environments with full protocol support (POSIX, NFS, SMB, S3, GDS).
- **Elastic scalability**: Dynamically grow or shrink performance and capacity to match your training and inference demands.
- **Certified performance**: NVIDIA BasePOD and SuperPOD validated, with proven performance at exabyte scale.
- **Built-in tiering**: Store active and cold data efficiently with transparent object-tiering for cost and energy savings.
- **Resilient and recoverable**: Snap-to-object capabilities enable fast, scalable recovery and hybrid data mobility.

Modern AI pipelines can’t afford slowdowns, stalls, or surprises. With NeuralMesh , you get infrastructure that’s built from the ground up for AI workloads—capable of handling aggressive checkpointing, mixed I/O, and unpredictable demands with ease. Whether you're running hundreds of experiments or scaling a production pipeline, NeuralMesh helps you move faster, recover smarter, and get the full value out of every GPU hour.

Ready to see how NeuralMesh performs in your environment?
Contact us to learn more.

Appendix

How To build a NeMo toolkit

Use the NeMo QuickStart guide at https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/main/starthere/intro.html to set up and deploy a basic NeMo environment.

Fine-tuning FastPitch with the NeMo toolkit

https://github.com/NVIDIA/NeMo/blob/stable/tutorials/tts/FastPitch_Finetuning.ipynb

FastPitch Training Setup and Parameters for the Test Run

```
[NeMo W 2024-02-02 21:47:39 nemo_logging:349]
/usr/local/lib/python3.10/dist-packages/hydra/_internal/hydra.py:1
19: UserWarning: Future Hydra versions will no longer change
working directory at job runtime by default.
See
https://hydra.cc/docs/1.2/upgrades/1.1_to_1.2/changes_to_job_worki
ng_dir/ for more information.
ret = run_job(
[NeMo W 2024-02-02 21:47:39 nemo_logging:349]
/usr/local/lib/python3.10/dist-packages/lightning_fabric/connector
.py:554: UserWarning: 16 is supported for historical reasons but
its usage is discouraged. Please set your precision to 16-mixed
instead!
rank_zero_warn(
Using 16bit Automatic Mixed Precision (AMP)
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
[NeMo I 2024-02-02 21:47:39 exp_manager:394] Experiments will be
logged at
ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-02_21-47-39
[NeMo I 2024-02-02 21:47:39 exp_manager:835] TensorboardLogger has
been set up
[NeMo W 2024-02-02 21:47:39 exp_manager:931] The checkpoint
callback was told to monitor a validation value and trainer's
max_steps was set to 1000. Please ensure that max_steps will run
for at least 25 epochs to ensure that checkpointing will not error
out.
NeMo-text-processing :: INFO :: Creating ClassifyFst grammars.
Creating ClassifyFst grammars.
```

```
[NeMo W 2024-02-02 21:48:09 en_us_arpabet:66]
apply_to_oov_word=None, This means that some of words will remain
unchanged if they are not handled by any of the rules in
self.parse_one_word(). This may be intended if phonemes and chars
are both valid inputs, otherwise, you may see unexpected deletions
in your input.
[NeMo I 2024-02-02 21:48:09 dataset:229] Loading dataset from
./9017_manifest_train_dur_5_mins_local.json.
76it [00:00, 2569.52it/s]
[NeMo I 2024-02-02 21:48:09 dataset:267] Loaded dataset with 76
files.
[NeMo I 2024-02-02 21:48:09 dataset:269] Dataset contains 0.08
hours.
[NeMo I 2024-02-02 21:48:09 dataset:377] Pruned 0 files. Final
dataset contains 76 files
[NeMo I 2024-02-02 21:48:09 dataset:379] Pruned 0.00 hours. Final
dataset contains 0.08 hours.
[NeMo I 2024-02-02 21:48:09 dataset:229] Loading dataset from
./9017_manifest_dev_ns_all_local.json.2it [00:00, 2545.86it/s]
[NeMo I 2024-02-02 21:48:09 dataset:267] Loaded dataset with 2
files.
[NeMo I 2024-02-02 21:48:09 dataset:269] Dataset contains 0.00
hours.
[NeMo I 2024-02-02 21:48:09 dataset:377] Pruned 0 files. Final
dataset contains 2 files
[NeMo I 2024-02-02 21:48:09 dataset:379] Pruned 0.00 hours. Final
dataset contains 0.00 hours.
[NeMo I 2024-02-02 21:48:09 features:289] PADDING: 1
NeMo-text-processing :: INFO :: Creating ClassifyFst grammars.
Creating ClassifyFst grammars.
[NeMo W 2024-02-02 21:48:42 en_us_arpabet:66]
apply_to_oov_word=None, This means that some of words will remain
unchanged if they are not handled by any of the rules in
self.parse_one_word(). This may be intended if phonemes and chars
are both valid inputs, otherwise, you may see unexpected deletions
in your input.
[NeMo W 2024-02-02 21:48:42 modelPT:161] If you intend to do
training or fine-tuning, please call the
ModelPT.setup_training_data() method and provide a valid
configuration file to setup the train data loader.
Train config :
dataset:
_target_: nemo.collections.tts.torch.data.TTSDataset
manifest_filepath:
/ws/LJSpeech/nvidia_ljspeech_train_clean_ngc.json
sample_rate: 22050
```

```
sup_data_path: /raid/LJSpeech/supplementary
sup_data_types:
- align_prior_matrix
- pitch
n_fft: 1024
win_length: 1024
hop_length: 256
window: hann
n_mels: 80
lowfreq: 0
highfreq: 8000
max_duration: null
min_duration: 0.1
ignore_file: null
trim: false
pitch_fmin: 65.40639132514966
pitch_fmax: 2093.004522404789
pitch_norm: true
pitch_mean: 212.35873413085938
pitch_std: 68.52806091308594
use_beta_binomial_interpolator: true
dataloader_params:
drop_last: false
shuffle: true
batch_size: 24
num_workers: 0

[NeMo W 2024-02-02 21:48:42 modelPT:168] If you intend to do
validation, please call the ModelPT.setup_validation_data() or
ModelPT.setup_multiple_validation_data() method and provide a
valid configuration file to setup the validation data loader(s).
Validation config :
dataset:
_target_: nemo.collections.tts.torch.data.TTSDataset
manifest_filepath: /ws/LJSpeech/nvidia_ljspeech_val_clean_ngc.json
sample_rate: 22050
sup_data_path: /raid/LJSpeech/supplementary
sup_data_types:
- align_prior_matrix
- pitch
n_fft: 1024
win_length: 1024
hop_length: 256
window: hann
n_mels: 80
lowfreq: 0
```

```
highfreq: 8000
max_duration: null
min_duration: null
ignore_file: null
trim: false
pitch_fmin: 65.40639132514966
pitch_fmax: 2093.004522404789
pitch_norm: true
pitch_mean: 212.35873413085938
pitch_std: 68.52806091308594
use_beta_binomial_interpolator: true
dataloader_params:
drop_last: false
shuffle: false
batch_size: 24
num_workers: 0

[NeMo I 2024-02-02 21:48:42 features:289] PADDING: 1
[NeMo I 2024-02-02 21:48:42 save_restore_connector:249] Model
FastPitchModel was successfully restored from
/NeMo/weka-workspace/nemo/tutorials/tts/tts_en_fastpitch_align.nem
o.
[NeMo I 2024-02-02 21:48:42 modelPT:1234] Model checkpoint
restored from nemo file with path :
`./tts_en_fastpitch_align.nemo`
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
[NeMo I 2024-02-02 21:48:43 modelPT:728] Optimizer config = Adam (
Parameter Group 0
amsgrad: False
betas: [0.9, 0.999]
capturable: False
differentiable: False
eps: 1e-08
foreach: None
fused: None
lr: 0.0002
maximize: False
weight_decay: 1e-06
)
[NeMo I 2024-02-02 21:48:43 lr_scheduler:772] Scheduler not
initialized as no `sched` config supplied to setup_optimizer()
  | Name                | Type                              |
Params
------------------------------------------------------------------
--------
0 | mel_loss_fn         | MelLoss                           | 0
```

```
1 | pitch_loss_fn       | PitchLoss                          | 0
2 | duration_loss_fn    | DurationLoss                       | 0
3 | energy_loss_fn      | EnergyLoss                         | 0
4 | aligner             | AlignmentEncoder                   | 1.0
M
5 | forward_sum_loss_fn | ForwardSumLoss                     | 0
6 | bin_loss_fn         | BinLoss                            | 0
7 | preprocessor        | AudioToMelSpectrogramPreprocessor | 0
8 | fastpitch           | FastPitchModule                    | 45.8
M
-------------------------------------------------------------------
--------
45.8 M Trainable params
0 Non-trainable params
45.8 M Total params
183.035 Total estimated model params size (MB)
[NeMo W 2024-02-02 21:48:48 nemo_logging:349]
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/loops/fi
t_loop.py:281: PossibleUserWarning: The number of training batches
(7) is smaller than the logging interval
Trainer(log_every_n_steps=100). Set a lower value for
log_every_n_steps if you want to see logs for the training epoch.
rank_zero_warn(
Training: 0it [00:00, ?it/s][NeMo I 2024-02-02 21:48:48
preemption:56] Preemption requires torch
distributed to be initialized, disabling preemption
Epoch 24: 100%|█| 7/7 [00:02<00:00, 3.21it/s, v_num=7-39,
train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation: 0%|                                    | 0/1 [00:00<?,
?it/s]
Validation DataLoader 0: 0%|                       | 0/1 [00:00<?,
?it/s]
Epoch 24: 100%|█| 7/7 [00:02<00:00, 2.71it/s, v_num=7-39,
train_step_timing in
                                                    Epoch 24, global
step 175: 'val_loss' reached 1.89732 (best 1.89732), saving model
to
'/NeMo/weka-workspace/nemo/tutorials/tts/ljspeech_to_9017_no_mixin
g_5_mins/FastPitch/2024-02-02_21-47-39/checkpoints/FastPitch--val_
loss=1.8973-epoch=24.ckpt' as top 3
Epoch 49: 100%|█| 7/7 [00:02<00:00, 3.18it/s, v_num=7-39,
train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation: 0%|                                    | 0/1 [00:00<?,
?it/s]
```

```
Validation DataLoader 0: 0%|                    | 0/1 [00:00<?,
?it/s]
Epoch 49: 100%|█| 7/7 [00:02<00:00, 2.65it/s, v_num=7-39,
train_step_timing in
                                                Epoch 49, global
step 350: 'val_loss' reached 2.04128 (best 1.89732), saving model
to
'/NeMo/weka-workspace/nemo/tutorials/tts/ljspeech_to_9017_no_mixin
g_5_mins/FastPitch/2024-02-02_21-47-39/checkpoints/FastPitch--val_
loss=2.0413-epoch=49.ckpt' as top 3
Epoch 74: 100%|█| 7/7 [00:02<00:00, 3.28it/s, v_num=7-39,
train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation: 0%|                                 | 0/1 [00:00<?,
?it/s]
Validation DataLoader 0: 0%|                    | 0/1 [00:00<?,
?it/s]
Epoch 74: 100%|█| 7/7 [00:02<00:00, 2.71it/s, v_num=7-39,
train_step_timing in
                                                Epoch 74, global
step 525: 'val_loss' reached 2.00175 (best 1.89732), saving model
to '/NeMo/weka-workspace/nemo/
tutorials/tts/ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-
02_21-47-39/checkpoints/FastPitch--
val_loss=2.0018-epoch=74.ckpt' as top 3
Epoch 99: 100%|█| 7/7 [00:02<00:00, 3.16it/s, v_num=7-39,
train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation: 0%|                                 | 0/1 [00:00<?,
?it/s]
Validation DataLoader 0: 0%|                    | 0/1 [00:00<?,
?it/s]
Epoch 99: 100%|█| 7/7 [00:02<00:00, 2.63it/s, v_num=7-39,
train_step_timing in
                                                Epoch 99, global
step 700: 'val_loss' reached 1.92470 (best 1.89732), saving model
to '/NeMo/weka-workspace/nemo/
tutorials/tts/ljspeech_to_9017_no_mixing_5_mins/FastPitch/2024-02-
02_21-47-39/checkpoints/FastPitch--
val_loss=1.9247-epoch=99.ckpt' as top 3
Epoch 124: 100%|█| 7/7 [00:02<00:00, 3.35it/s, v_num=7-39,
train_step_timing in
Validation: 0it [00:00, ?it/s]
Validation: 0%|                                 | 0/1 [00:00<?,
?it/s]
```

```
Validation DataLoader 0: 0%|                    | 0/1 [00:00<?,
?it/s]
Epoch 124: 100%|█| 7/7 [00:02<00:00, 2.75it/s, v_num=7-39,
train_step_timing in
                                                Epoch 124, global
step 875: 'val_loss' reached 1.86749 (best 1.86749), saving model
to
'/NeMo/weka-workspace/nemo/tutorials/tts/ljspeech_to_9017_no_mixin
g_5_mins/FastPitch/2024-02-02_21-47-39/checkpoints/FastPitch--val_
loss=1.8675-epoch=124.ckpt' as top 3
```