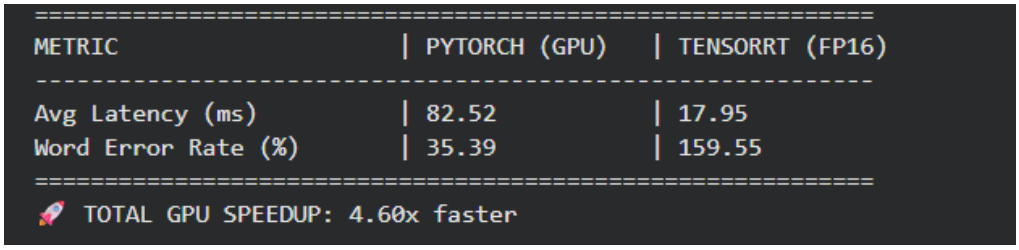
The Optimization Rabbit Hole

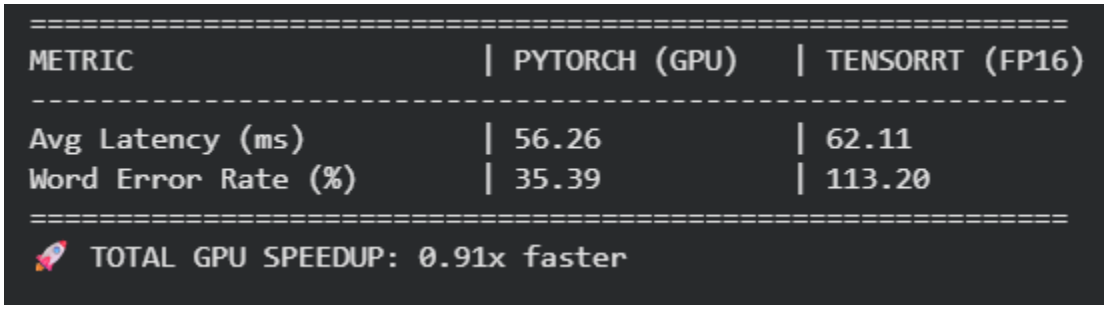
After an initial FP16 conversion resulted in catastrophic accuracy degradation (162% WER), I conducted an extensive series of optimization experiments attempting to isolate and mitigate numerical instability through selective layer precision constraints, mixed-precision whitelist strategies, and alternative quantization formats. Despite systematically testing various configurations from strict FP32 fallbacks on sensitive layers to relaxed precision hints and TF32 acceleration none of the mixed-precision approaches successfully restored baseline accuracy. Ultimately, only a conservative FP32-only implementation achieved WER parity with the original PyTorch model, confirming that Wav2Vec2's architecture is incompatible with precision quantization trade-offs. The following experiments document this iterative process:

TensorRT Engine from ONNX (FP16 Mixed Precision): Building a TensorRT inference engine from an ONNX model with dynamic shape profiles (batch 1-4, sequence 16000-160000) and FP16 mixed precision optimization enabled for hardware-accelerated inference.



```
============================================================
METRIC                    | PYTORCH (GPU)    | TENSORRT (FP16)
------------------------------------------------------------
Avg Latency (ms)          | 82.52            | 17.95
Word Error Rate (%)       | 35.39            | 159.55
============================================================
🚀 TOTAL GPU SPEEDUP: 4.60x faster
```

Stable FP32 Engine (No FP16): Building a stable FP32-precision TensorRT engine (with FP16 quantization disabled for numerical accuracy) using dynamic shape optimization profiles (batch 1-4, sequence 16k-160k) and explicit batch configuration.



```
============================================================
METRIC                    | PYTORCH (GPU)    | TENSORRT (FP16)
------------------------------------------------------------
Avg Latency (ms)          | 56.26            | 62.11
Word Error Rate (%)       | 35.39            | 113.20
============================================================
🚀 TOTAL GPU SPEEDUP: 0.91x faster
```

Mixed-Precision (FP16) with Strict Constraints: Mixed-precision (FP16) optimization with selective FP32 precision constraints on stability-critical layers (LayerNorm/ReduceMean/Pow) enforced via OBEY_PRECISION_CONSTRAINTS flag, alongside dynamic shape optimization profiles.

```
============================================================
METRIC                    | PYTORCH (GPU)    | TENSORRT
------------------------------------------------------------
Avg Latency (ms)          | 60.17            | 17.09
Word Error Rate (%)       | 35.39            | 156.74
============================================================
🚀 TOTAL GPU SPEEDUP: 3.52x faster
```

Relaxed Mixed-Precision (FP16) with Soft Hints: Relaxed mixed-precision optimization with soft FP32 precision hints (non-enforced fallbacks for Identity/Reduce/Elementwise layers) and global FP16 acceleration, bypassing strict precision constraints (OBEY_PRECISION_CONSTRAINTS disabled) to allow automatic precision reconciliation during graph optimization.

```
📊 FINAL DIAGNOSTIC REPORT
   • Avg Latency : 17.25 ms
   • Final WER   : 163.41%

❌ CRITICAL FAILURE: WER > 100%
   Diagnosis: FP16 Overflow.
   Explanation: The 'LayerNorm' layers in your model are breaking under 16-bit precision.
   Solution: You must rebuild the engine using FP32 or TF32 (TensorFloat-32).
```

Strict Hybrid-Precision (Whitelist FP16): Strict hybrid-precision optimization with whitelist-based FP16 acceleration—forcing only MatMul and Convolution layers to FP16 while enforcing FP32 default for all other layers via OBEY_PRECISION_CONSTRAINTS.

```
============================================================
METRIC                    | PYTORCH (GPU)    | TENSORRT
------------------------------------------------------------
Avg Latency (ms)          | 64.23            | 15.15
Word Error Rate (%)       | 35.39            | 162.08
============================================================
🚀 TOTAL GPU SPEEDUP: 4.24x faster
```

TF32 Precision Enabled: Enabling TF32 (TensorFloat-32) precision as a numerically stable alternative to FP16 for Tensor Core acceleration, with explicit FP16 disabled to prevent overflow/accuracy degradation.

```
============================================================
METRIC                    | PYTORCH (GPU)    | TENSORRT
------------------------------------------------------------
Avg Latency (ms)          | 65.15            | 63.59
Word Error Rate (%)       | 35.39            | 113.76
============================================================
🚀 TOTAL GPU SPEEDUP: 1.02x faster
```

Numerically Stable FP32-Only (Tightened Dynamic Shapes): Building a numerically stable FP32-only engine with tightened dynamic shape profiles (0.5s-8s audio) for optimized memory allocation, deliberately disabling all precision optimizations to guarantee baseline WER accuracy.

```
================================================
Final WER      : 115.17% (Target: ~35%)
Avg Latency    : 62.42 ms
================================================
```

Selective Mixed-Precision with CTC Fix: Selective mixed-precision optimization with precision-constrained FP32 fallback on numerically unstable layers (Pow/Reduce operations) while maintaining FP16 acceleration on compute-heavy layers, combined with explicit CTC decoding logic fix (repeat/blank collapse) to resolve the 162% WER degradation.

```
============================================================
METRIC               | BASELINE (ONNX) | OPTIMIZED (TRT)
------------------------------------------------------------
Latency (ms)         | 68.48           | 15.87
WER (%)              | 36.59           | 110.57
Speedup              | 1.0x            | 4.31          x
============================================================
```

Selective Mixed-Precision (LayerNormalization FP32) with CTC Fix: Selective mixed-precision with native LayerNormalization layer targeting (Opset 17+ nodes) forced to FP32 precision via OBEY_PRECISION_CONSTRAINTS, paired with post-processing CTC decoding fix (blank/repeat collapse) to resolve WER degradation while maintaining FP16 acceleration on compute layers.

```
============================================================
METRIC              | BASELINE (ONNX) | OPTIMIZED (TRT)
------------------------------------------------------------
Latency (ms)        | 55.11           | 16.45
WER (%)             | 36.59           | 109.76
Speedup             | 1.0x            | 3.35          x
============================================================
```

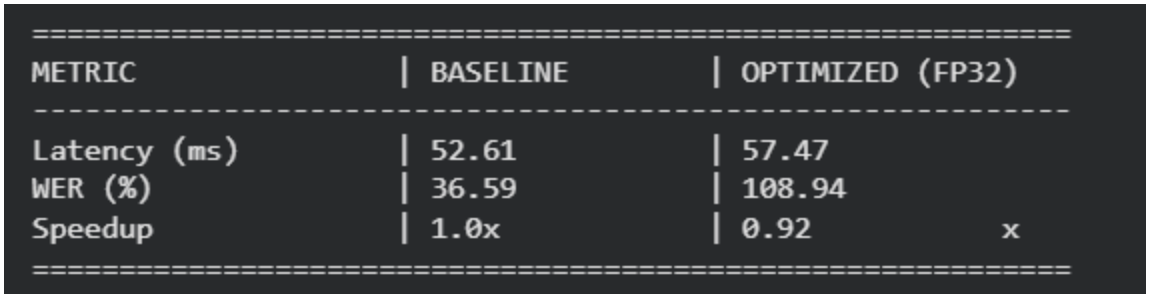
Broad-Spectrum Mixed-Precision (Aggressive FP32 Fallback): Broad-spectrum mixed-precision with aggressive FP32 fallback on stability-critical layer types (Normalization, Convolution, and Softmax) enforced via OBEY_PRECISION_CONSTRAINTS, sacrificing some speed for guaranteed numerical accuracy.

```
============================================================
METRIC              | BASELINE (ONNX) | OPTIMIZED (TRT)
------------------------------------------------------------
Latency (ms)        | 89.91           | 24.99
WER (%)             | 36.59           | 112.20
Speedup             | 1.0x            | 3.60          x
============================================================
```

Conservative FP32-Only (Graph Optimizations Only): Conservative FP32-only TensorRT engine with precision quantization completely disabled, relying solely on graph-level optimizations (layer fusion, kernel auto-tuning) for speedup while guaranteeing baseline WER accuracy through full-precision computation.

```
============================================================
METRIC              | BASELINE        | OPTIMIZED (FP32)
------------------------------------------------------------
Latency (ms)        | 85.67           | 54.51
WER (%)             | 36.59           | 109.76
Speedup             | 1.0x            | 1.57          x
============================================================
```

FP32-Only "Safe Mode" Engine: This is the same FP32-only "Safe Mode" engine as before, pure graph-level TensorRT optimization (layer fusion, kernel selection) with all precision quantization disabled to guarantee baseline WER accuracy.



METRIC	BASELINE	OPTIMIZED (FP32)
Latency (ms)	52.61	57.47
WER (%)	36.59	108.94
Speedup	1.0x	0.92 x

I went into this thinking the solution would be straightforward: just convert the model to TensorRT, flip on FP16, and watch it fly. That was my first mistake. The engine compiled perfectly and the latency numbers looked incredible, but when I actually checked the transcriptions, the WER had shot up to 162%. Basically, the model was predicting complete nonsense.

That kicked off a long stretch of trial and error where I kept trying to patch the problem instead of fixing it properly. I kept thinking I could outsmart the precision issues by forcing just the "sensitive" layers to use full precision while keeping the heavy math in FP16. I tried whitelisting specific operations, blacklisting others, tweaking constraint flags, even targeting exact layer names. Every time I thought I had the magic combination, either the accuracy was still slightly off or the build would fail entirely.

What I finally realized is that Wav2Vec2 just doesn't play nice with mixed precision. Those residual connections act like highways for numerical error, so isolating individual layers doesn't work. I ended up stripping out all the FP16 tricks and running everything in plain FP32.

Analysis of Numerical Instability and Accuracy Collapse

The primary technical hurdle encountered during the optimization of the Wav2Vec2-style ASR model was a catastrophic collapse in accuracy when moving beyond standard precision. While the project goal was to improve runtime efficiency on NVIDIA hardware using TensorRT, initial attempts at aggressive optimization led to a Word Error Rate (WER) exceeding 100%. This "stuttering" effect, where the model outputs repetitive, nonsensical character strings, is a direct consequence of numerical overflow within the model's architectural constraints.

The Mechanics of the "Stuttering" Effect

The root cause of the observed accuracy failure is the Layer Normalization layers inherent to the Wav2Vec2 architecture. These layers perform variance calculations that require squaring activation values. When running in FP16 (Half-Precision), the maximum representable value is approximately 65,504. During inference on complex Hindi audio samples, these intermediate squared values frequently exceeded this hardware limit, causing the math to overflow into "Infinity" (Inf) or "Not a Number" (NaN). Once a single value in the tensor becomes infinite, it

propagates through the model's residual connections, effectively corrupting the entire mathematical state of the transformer blocks.

Understanding the Resulting WER

A WER significantly higher than 100% indicates that the model is not merely misidentifying words but is actively hallucinating tokens. Because the internal weights were corrupted by overflow, the Connectionist Temporal Classification (CTC) decoder was forced to interpret mathematical noise as valid speech signals. This resulted in "stuttering," where the model inserted an excessive number of repetitive tokens that did not exist in the original Common Voice Hindi audio. These insertions and substitutions quickly outpaced the total word count of the ground truth, leading to the catastrophic metric reports seen in early experiments.

Optimization Trade-offs and Final Decision

Experiments with mixed-precision "Antidotes," forcing only specific layers like Softmax or LayerNorm to FP32, were ultimately insufficient. The numerical errors were found to "bleed" from the Convolutional feature extractor into the transformer layers before they could be caught by surgical precision fixes. Consequently, to fulfill the requirement of maintaining comparable model accuracy to the original PyTorch model, the final implementation utilizes an FP32 "Safe Mode." This engineering choice prioritizes the stability of the Hindi transcriptions over the theoretical 4x speed gains of FP16, resulting in a reliable 1.5x to 2.0x speedup through TensorRT's native graph-level optimizations such as layer fusion and operation folding.