**3GPP TSG-SA WG4 Meeting #Audio SWG AH S4aA250203**

**Erlangen, 23 – 25 Sept. 2025**

**Source: vivo, MediaTek Inc., Bytedance, Spreadtrum, CMCC, Fraunhofer IIS**

**Title: [FS\_ULBC] Discussion on CPU-centric Complexity for ULBC**

**Agenda item: 4.4**

**Document for: DISCUSSION and AGREEMENT**

**1. Introduction**

The ongoing study on the Ultra-Low Bitrate Codec (ULBC) [1] has initiated critical discussions on defining appropriate complexity constraints for next-generation, AI-driven speech codecs. Current discussions [2] have heavily focused on metrics related to Neural Processing Units (NPUs), such as TOPS and TOPS/W, positioning the NPU as the primary target for deployment.

However, this contribution argues for a revised perspective. A deep analysis of Google state-of-the-art Lyra V2 [3] codec reveals that its remarkable performance is achieved entirely on the CPU. This verifiable, real-world example demonstrates that highly efficient, state-of-the-art AI codecs can be deployed successfully without depending on the fragmented and unpredictable NPU ecosystem.

This paper proposes that CPU performance should be considered a primary design objective for the ULBC, not merely a fallback. We provide a detailed analysis of the Lyra V2 architecture to support this position and suggest capturing these findings within the technical report TR 26.940 to ensure the final ULBC standard is accessible, reliable, and performs consistently across the widest possible range of devices.

**2. Analysis of an existing AI codec: Lyra V2**

Google Lyra V2 codec serves as a powerful case study. Its publicly reported performance of "38x faster than real-time" on a Pixel 6 Pro [4] is achieved entirely via CPU execution. A direct analysis of the open-source codebase confirms that all performance-critical components are implemented to run directly on the CPU, with a complete absence of NPU/TPU delegate calls. Furthermore, the codec is open-sourced under the Apache 2.0 license [5], which is permissive for commercial and standardization purposes, making it an ideal reference for this study.

**2.1. Code-Level Analysis**

The Lyra V2 implementation consistently defaults to a CPU-based execution path for neural network inference.

* The core *LyraGanModel* [6] and *SoundStreamEncoder* [7] components explicitly use a flag (**use\_xnn=true**) that directs the TensorFlow Lite [9] interpreter to use its CPU backend (XNNPACK backend [8]) for execution, rather than offloading to specialized hardware accelerators.

|  |
| --- |
| [Location: lyra/lyra\_gan\_model.cc](https://github.com/google/lyra/blob/47698dadf0010abff6a848e02642f55f806d4842/lyra/lyra_gan_model.cc#L36) std::unique\_ptr<LyraGanModel> LyraGanModel::Create(     const ghc::filesystem::path& model\_path, int num\_features) {   **auto** model =       TfLiteModelWrapper::Create(model\_path / "lyragan.tflite",                                  */\*use\_xnn=\*/***true**, */\*int8\_quantized=\*/***true**);   **if** (model == **nullptr**) {     LOG(ERROR) << "Unable to create LyraGAN TFLite model wrapper.";     **return** **nullptr**;   } [Location: lyra/soundstream\_encoder.cc](https://github.com/google/lyra/blob/47698dadf0010abff6a848e02642f55f806d4842/lyra/soundstream_encoder.cc#L36) std::unique\_ptr<SoundStreamEncoder> SoundStreamEncoder::Create(     const ghc::filesystem::path& model\_path) {   **auto** model = TfLiteModelWrapper::Create(       model\_path / "soundstream\_encoder.tflite", */\*use\_xnn=\*/***true**,       */\*int8\_quantized=\*/***false**);   **if** (model == **nullptr**) {     LOG(ERROR) << "Unable to create SoundStream TFLite model wrapper.";     **return** **nullptr**;   } |

* The TFLite model wrapper implementation is notably devoid of any delegates for hardware acceleration. There are no calls to the NNAPI [10] delegate (for Android NPUs), Hexagon delegate [11] (for Qualcomm DSPs), CoreML [12] delegate (for Apple Neural Engine), or TPU [13] delegate. The architecture is fundamentally CPU-centric.
* It should be noted that the number of threads is explicitly set to only 1 in the Lyra code, as can be seen here in [lyra/lyra/tflite\_model\_wrapper.cc Line 68](https://github.com/google/lyra/blob/main/lyra/tflite_model_wrapper.cc#L68C5-L72C43) (*https://github.com/google/lyra/blob/main/lyra/tflite\_model\_wrapper.cc#L68C5-L72C43*).
* The benchmark results provided in the codebase comments confirm exceptional performance (38x real-time on a Pixel 6 Pro [4]) achieved through this CPU-only pathway.

|  |
| --- |
| [Location: Comments showing Pixel 6 Pro results](https://github.com/google/lyra/blob/47698dadf0010abff6a848e02642f55f806d4842/README.md?plain=1#L153) *# If you press 'Benchmark', you should see something like the following in logcat*  *# on a Pixel 6 Pro when running the benchmark:*  lyra\_benchmark:  feature\_extractor:  max: 1.836 ms  min: 0.132 ms  mean: 0.153 ms lyra\_benchmark: quantizer\_quantize:  max: 1.042 ms  min: 0.120 ms  mean: 0.130 ms lyra\_benchmark:   quantizer\_decode:  max: 0.103 ms  min: 0.026 ms  mean: 0.029 ms lyra\_benchmark:       model\_decode:  max: 0.820 ms  min: 0.191 ms  mean: 0.212 ms lyra\_benchmark:              total:  max: 2.536 ms  min: 0.471 ms  mean: 0.525 ms |

A mean processing time of 0.525 ms for a 20 ms audio frame corresponds to being ~38 times faster than real-time.

**2.2. Implications of the Lyra V2 Architecture**

The success of Lyra V2 CPU-only approach provides several critical insights for the ULBC study:

1. It definitively proves that a state-of-the-art, low-bitrate AI speech codec can achieve and exceed real-time performance requirements on modern smartphone CPUs with a significant margin towards max. RTF.
2. Evaluating CPU performance ensures that the ULBC will function efficiently on all devices. This aligns with the goal of creating a universal standard.

**3. Proposals for TR 26.940**

Based on the conclusive evidence from the Lyra V2 analysis, it is proposed that the following changes be made in TR 26.940 to guide the definition of design constraints:

\* \* \* First Change \* \* \* \*

# 7 Existing technologies and feasibility evidence

Editor’s Note:

1. Provide some evidence that the design criteria can be met, for example existing reference codecs.

## 7.2 Analysis of an existing AI codec: Lyra V2

Google Lyra V2 codec serves as a powerful case study. Its publicly reported performance of "38x faster than real-time" on a Pixel 6 Pro [4] is achieved entirely via CPU execution. A direct analysis of the open-source codebase confirms that all performance-critical components are implemented to run directly on the CPU, with a complete absence of NPU/TPU delegate calls. Furthermore, the codec is open-sourced under the Apache 2.0 license [5], which is permissive for commercial and standardization purposes, making it an ideal reference for this study.

7.2.1. Code-Level analysis

The Lyra V2 implementation consistently defaults to a CPU-based execution path for neural network inference.

* The core *LyraGanModel* [6] and *SoundStreamEncoder* [7] components explicitly use a flag (**use\_xnn=true**) that directs the TensorFlow Lite [9] interpreter to use its CPU backend (XNNPACK backend [8]) for execution, rather than offloading to specialized hardware accelerators.

|  |
| --- |
| [Location: lyra/lyra\_gan\_model.cc](https://github.com/google/lyra/blob/47698dadf0010abff6a848e02642f55f806d4842/lyra/lyra_gan_model.cc#L36) std::unique\_ptr<LyraGanModel> LyraGanModel::Create(     const ghc::filesystem::path& model\_path, int num\_features) {   **auto** model =       TfLiteModelWrapper::Create(model\_path / "lyragan.tflite",                                  */\*use\_xnn=\*/***true**, */\*int8\_quantized=\*/***true**);   **if** (model == **nullptr**) {     LOG(ERROR) << "Unable to create LyraGAN TFLite model wrapper.";     **return** **nullptr**;   } [Location: lyra/soundstream\_encoder.cc](https://github.com/google/lyra/blob/47698dadf0010abff6a848e02642f55f806d4842/lyra/soundstream_encoder.cc#L36) std::unique\_ptr<SoundStreamEncoder> SoundStreamEncoder::Create(     const ghc::filesystem::path& model\_path) {   **auto** model = TfLiteModelWrapper::Create(       model\_path / "soundstream\_encoder.tflite", */\*use\_xnn=\*/***true**,       */\*int8\_quantized=\*/***false**);   **if** (model == **nullptr**) {     LOG(ERROR) << "Unable to create SoundStream TFLite model wrapper.";     **return** **nullptr**;   } |

* The TFLite model wrapper implementation is notably devoid of any delegates for hardware acceleration. There are no calls to the NNAPI [10] delegate (for Android NPUs), Hexagon delegate [11] (for Qualcomm DSPs), CoreML [12] delegate (for Apple Neural Engine), or TPU [13] delegate. The architecture is fundamentally CPU-centric.
* It should be noted that the number of threads is explicitly set to only 1 in the Lyra code, as can be seen here in [lyra/lyra/tflite\_model\_wrapper.cc Line 68](https://github.com/google/lyra/blob/main/lyra/tflite_model_wrapper.cc#L68C5-L72C43) (*https://github.com/google/lyra/blob/main/lyra/tflite\_model\_wrapper.cc#L68C5-L72C43*).
* The benchmark results provided in the codebase comments confirm exceptional performance (38x real-time on a Pixel 6 Pro [4]) achieved through this CPU-only pathway.

|  |
| --- |
| [Location: Comments showing Pixel 6 Pro results](https://github.com/google/lyra/blob/47698dadf0010abff6a848e02642f55f806d4842/README.md?plain=1#L153) *# If you press 'Benchmark', you should see something like the following in logcat*  *# on a Pixel 6 Pro when running the benchmark:*  lyra\_benchmark:  feature\_extractor:  max: 1.836 ms  min: 0.132 ms  mean: 0.153 ms lyra\_benchmark: quantizer\_quantize:  max: 1.042 ms  min: 0.120 ms  mean: 0.130 ms lyra\_benchmark:   quantizer\_decode:  max: 0.103 ms  min: 0.026 ms  mean: 0.029 ms lyra\_benchmark:       model\_decode:  max: 0.820 ms  min: 0.191 ms  mean: 0.212 ms lyra\_benchmark:              total:  max: 2.536 ms  min: 0.471 ms  mean: 0.525 ms |

A mean processing time of 0.525 ms for a 20 ms audio frame corresponds to being ~38 times faster than real-time.

7.2.2. Conclusion

The success of Lyra V2 CPU-only approach provides several critical insights for the ULBC study and definitively proves that a state-of-the-art, low-bitrate AI speech codec, similar to Lyra v2, can achieve and exceed real-time performance requirements on a modern smartphone CPU, such as Pixel 6 Pro, with a significant margin towards max. RTF.

\* \* \* Second Change \* \* \* \*

Table 6.2-1 List of ULBC design constraint parameter

| Parameter | Design Constraint | Note |
| --- | --- | --- |
| Bit rates |  |  |
| Sample rate and audio bandwidth |  |  |
| Frame length |  |  |
| Complexity and memory demands |  | Clause 7 provides several complexity analyses for informational and reference purposes. |
| Algorithmic delay |  | The algorithmic delay is defined as the frame size buffering delay plus any other delays inherent in the codec algorithm (e.g., look-ahead, sample-rate conversion, and decoder post-processing) |
| Packet loss concealment (PLC) |  |  |
| Potential use of noise suppression as part of the codec |  |  |
| Discontinuous transmission including voice activity detection and comfort noise |  |  |
| Robustness to non-speech input |  | Editor’s note: May need to be in performance requirement |
|  |  |  |

\* \* \* Third Change \* \* \* \*

## 6.3 Design Constraint Verification

Editor’s note: Algorithmic delay verification method for AI based codecs required.

### 6.3.1 Complexity Verification

Given the evidence that state-of-the-art AI codecs can run efficiently on the CPU, it is proposed that the ULBC study treat CPU performance (such as RTF and RTF distributions across a set of representative devices) as a metric of codec complexity verification.

\* \* \* End of Changes \* \* \* \*

**Reference**

[1] SP-250378 "New SID on Ultra Low Bitrate Speech Codec".

[2] SP-251326 "On ULBC complexity design constraints".

[3] Lyra V2 - a better, faster, and more versatile speech codec [Internet]. Google Open Source Blog; 2022 Sept 30 [cited 2025 Aug 15]. Available from: <https://opensource.google.com/blog>

[4] Google. Lyra: A generative low bitrate speech codec [Internet]. GitHub; [cited 2025 Aug 15]. Available from: <https://github.com/google/lyra/blob/main/README.md#building-for-android>

[5] Apache License, Version 2.0 [Internet]. Apache Software Foundation; 2004 Jan [cited 2025 Aug 15]. Available from: <https://github.com/google/lyra?tab=Apache-2.0-1-ov-file>

[6] Google, lyra\_gan\_model.cc, Lyra repository, commit 47698da, viewed 15 August 2025, <https://github.com/google/lyra/blob/47698dadf0010abff6a848e02642f55f806d4842/lyra/lyra_gan_model.cc#L36>

[7] Google, soundstream\_encoder.cc, Lyra repository, commit 47698da, viewed 15 August 2025, <https://github.com/google/lyra/blob/47698dadf0010abff6a848e02642f55f806d4842/lyra/soundstream_encoder.cc#L36>

[8] Google. XNNPACK: High-efficiency floating-point neural network inference operators for mobile, server, and Web [Internet]. GitHub; [cited 2025 Aug 15]. Available from: <https://github.com/google/XNNPACK>

[9] Google AI for Developers. LiteRT overview [Internet]. Google AI Edge; [updated 2025 May 19; cited 2025 Aug 15]. Available from: <https://ai.google.dev/edge/litert>

[10] ONNX Runtime. "NNAPI Execution Provider." ONNX Runtime Documentation. [Online]. Available: <https://onnxruntime.ai/docs/execution-providers/NNAPI-ExecutionProvider.html>

[11] Qualcomm, "QNN," in AI Engine Direct SDK, Jul. 07, 2025. [Online]. Available: <https://docs.qualcomm.com/bundle/publicresource/topics/80-63442-50/introduction.html>

[12] "Apple's Neural Engine (ANE)" Apple Wiki, Fandom. [Online]. Available: <https://apple.fandom.com/wiki/Neural_Engine>

[13] Google Cloud. Introduction to Cloud TPU [Internet]. Google Cloud Documentation; [updated 2025 Aug 11; cited 2025 Aug 15]. Available from: <https://cloud.google.com/tpu/docs/intro-to-tpu>