**3GPP TSG-SA WG4 Meeting #Audio SWG AH S4aA250223**

**Erlangen, 23 – 25 Sept. 2025 was S4-251333 was** **S4aA250202**

**Source: vivo, Bytedance, Spreadtrum**

**Title: [FS\_ULBC] Considerations on measuring ULBC complexity**

**Agenda item: 4.4**

**Document for: DISCUSSION and AGREEMENT**

### 1. Introduction

For the standardization of the new ULBC [1] codec, establishing a fair and relevant method for evaluating complexity is essential. While traditional metrics like WMOPS [2] are less applicable to modern AI-driven codecs, theoretical metrics such as Multiply-Accumulate Operations (MACs) or Floating Point Operations (FLOPs) offer a useful starting point for understanding a model's computational load.

However, as we move from theory to practice, it is worthwhile to consider how these theoretical figures translate to on-device performance. A potential gap can emerge between a model's FLOPs count and its actual execution speed, particularly given the diverse and fragmented hardware ecosystem of mobile devices. For instance, reliance on specific NPU capabilities for acceleration raises questions about consistent performance on devices that may lack such support, necessitating a fallback to CPU execution.

To better understand this relationship, this paper looks at a real-world example of an AI audio codec. We tested how it runs on both a computer and a mobile phone. The results show that there can be a big and surprising difference between the complexity numbers on paper and how the codec actually performs on real devices. Because of this, we think it could be helpful for the ULBC standard to consider real-world test results when defining complexity rules. This could help make sure the codec works well and provides a good experience for all users.

### 2. Considerations on AI Codec Complexity

When thinking about the complexity of AI codecs, it is helpful to look at how they work on real mobile phones. A model's theoretical FLOPs count is a good first guess of its computational complexity. However, this number alone might not show the full picture of how it will actually perform or how much battery it will use on a real device.

A codec's on-device performance is often affected by many other things. These can include the speed of the device's memory, the specific AI operations that the hardware (NPU) supports, and how well the software tools work [3]. As our analysis later will show, these factors can make a big difference.

The mobile AI hardware landscape is also quite diverse. Major companies like Qualcomm [4], MediaTek [5], HiSilicon [6], and Apple [7] all make different types of NPUs with their own software. This means a codec that is designed to run fast on one type of NPU might not run as well on another. It is also worth considering that the ULBC codec may need to be used on a wide range of devices, including more entry-level phones that may not have a dedicated NPU or have very limited NPU capabilities.

For all these reasons, ensuring a good and consistent experience for every user is important. This is why it is valuable to also test a codec's performance on a standard CPU (e.g., Armv8). For devices without a proper NPU, the CPU performance is not just a fallback, it is the primary mode of operation. Therefore, making sure the codec is runnable on a variety of CPUs of different levels of mobile devices helps ensure that the codec is widely accessible. The following sections will explore this idea with concrete test data.

### 3. Complexity Analysis of an existing AI Codec

To investigate the relationship between theoretical complexity and practical performance, we conducted a detailed analysis of a publicly available AI codec, DAC (Descriptive Audio Codec) [8]. For this analysis, we used the methodology described in Section 3.1 and a pretrained model from [11], which has a 44.1 kHz sample rate and an 8-kbps codec bitrate.

**3.1. Methodology for Complexity Analysis**

To provide objective and reproducible data, a standardized benchmarking methodology was employed. The methodology involves using the ONNX Runtime library [3] to execute AI models on target hardware. This approach allows for complexity performance measurement across different "execution providers." For this analysis, we used the standard CPU backend and NNAPI (Neural Networks API) [3] backend, which is Android's interface for offloading AI workloads to specialized hardware accelerators like NPUs.

For all tests, the original, unmodified pretrained model from reference [11] was used. No changes were made to the model architecture or its parameters, and no retraining was performed. The model's fully convolutional architecture allows it to process variable-length inputs; the different frame sizes tested were achieved simply by varying the length of the input audio sample provided to the model. Furthermore, no quantization (e.g., to INT8 or INT16) was applied. The original float model was used for all execution providers, including NNAPI, as quantizing this type of generative audio model can be challenging due to its sensitivity to precision.

The method is designed to measure key complexity indicators by processing audio frames of various durations. For each test configuration, warm-up iterations are performed before collecting detailed statistics over multiple runs. The primary metric gathered is the Real-Time Factor (RTF), calculated for the full end-to-end pipeline as well as for individual components (e.g., encoder and decoder stages). This allows for a comprehensive analysis of both overall complexity performance and potential bottlenecks within the model architecture.

**3.2. Theoretical Complexity Analysis**

The theoretical computational load of the DAC model was analyzed using two established profiling libraries, ptflops (v0.7.5) and thop (v2.0.17, via the ultralytics fork) [9][10], to cross-verify the results. The complexity scales with the audio frame size, increasing from approximately 1.4 GFLOP counts for a 20ms frame to 31.6 GFLOP counts for a 320ms frame, as shown in Figure 1.



Figure 1: DAC Model Theoretical Complexity Comparison

The model's complexity was profiled both end-to-end and by evaluating its components separately. To simulate real-world usage, the encoder was profiled using dummy input tensors representing raw audio waveforms of shape *[1, 1, floor(sample\_rate \* duration\_in\_seconds)]*, with the duration varied to measure complexity scaling. The decoder, which takes the quantized latent representation as input, was profiled using dummy tensors of shape *[1, 1024, T]*, where *T* (time) was varied using values *[1, 3, 6, 13, 27]* to correspond to increasing audio durations.

Initial analysis (as documented in S4-251333) was conducted using older library versions (ptflops v0.7.4 and thop v1.1.1), which produced aligned results. However, when re-evaluating with the current, updated library versions cited in this document, a different set of results was obtained. While the newer libraries are also aligned with each other, their figures do not match the previous analysis. The change was traced back to a fundamental shift in the calculation methodology for *ConvTranspose1d* layers between the older and newer generations of the profiling tools.

The complexity metric we are using here (GFLOP counts per frame) represents the total number of floating-point operations required to process a single input frame of a given length. The results from both tools were highly consistent. The model has 76.9M parameters, resulting in a model size of 293 MB.

**3.3. Real-World Inference Performance Analysis**

To see how the DAC model performs in the real world, we tested it on two very different types of devices. We chose a powerful high-end desktop computer to see how fast it could run in an ideal case. We also tested it on a high-end mobile phone, which is closer to what users will actually have. This helps us understand the performance gap between a best-case scenario and a typical use case.

For our main measurement, we used the Real-Time Factor (RTF). This is an important metric because an RTF lower than 1.0 means the codec can process audio in real-time, like during a phone call. A lower RTF is always better.

**3.4. Key Findings**

Our tests gave us some important insights into how the codec performs in the real world. These findings are shown in the figures below and help explain why just looking at theoretical numbers isn't enough.

* On the high-end desktop CPU (frequency fixed to 5.7GHz), the codec is not real-time capable with a single thread (RTF between 1.6 and 1.9). As shown in Figure 2, real-time performance (RTF between 0.67 and 0.86) is only achievable with multi-threaded execution (4 threads). However, it is still very slow for such a high-end desktop CPU.



Figure 2: DAC Model on AMD Ryzen 9 7950X Inference Performance Analysis

* On the high-end mobile SoC (Qualcomm Snapdragon 8 Gen 2), no tested configuration achieves real-time performance. The best-case mobile RTF was 2.125 (over 2x slower than real-time), with the worst case reaching 5.884 (nearly 6x slower than real-time).

We had hoped that using the phone's NPU through the NNAPI backend would make the codec faster. However, the results were inconsistent (Figure 3). Sometimes it helped a little, but for one test, it actually made performance much worse than using the CPU. This shows that we cannot simply assume that using an NPU will automatically lead to better performance.



Figure 3: DAC Model on Qualcomm Snapdragon 8 Gen 2 Inference Performance Analysis

* The most important finding is the large gap between the theoretical complexity (GFLOP counts per frame) and the actual measured performance (RTF), as shown in Figure 4. A model that seems reasonably efficient on paper (~2-5 GFLOP counts per frame) was not able to run in real-time on a top-of-the-line mobile phone. This result is the key reason why this analysis suggests that real-world testing is so important.



Figure 4: DAC Model Theoretical Complexity vs. Real-World Inference Performance

### 4. Conclusions

The analysis in this paper used the Real-Time Factor (RTF) as a practical tool to measure on-device performance and highlight the challenge of defining a fair complexity constraint. Subsequent discussions have correctly pointed out that any practical metric, including RTF, is highly dependent on specific platforms and optimizations. This dependency makes it difficult to use such metrics directly as a normative requirement without risking bias towards certain implementations.

Therefore, a more balanced approach may be beneficial. It is concluded that the primary complexity constraints could be based on theoretical metrics, such as MACs/FLOPs for AI-based components and WMOPS for traditional signal processing components, as these provide a platform-agnostic measure of algorithmic complexity. However, to ensure these theoretical numbers are meaningful for real-world deployment, it is essential that they are complemented by a verification process. This aligns with the principle of Design Constraint Verification. It is worth noting that other standards bodies, such as JPEG AI [12], have faced similar challenges with hardware-dependent performance, and their approach is to verification on common reference platforms to validate feasibility. This (i.e., JPEG AI approach) may offer valuable insights for this work.

The following observations support this conclusion:

* A significant gap can exist between theoretical complexity metrics and actual on-device performance. This justifies the need to verify theoretical claims with practical measurements.
* This performance gap is attributed to system-level factors not captured by theoretical counts, including the fragmented NPU hardware ecosystem, memory bandwidth bottlenecks, and the fact that NPUs are shared system resources not fully dedicated to the codec.
* A verification framework should consider the wide range of target devices. This includes different classes of phones (e.g., high-end, low-end) and those that may lack a capable NPU, which would help ensure universal accessibility and a fair evaluation.

### 5. Proposal

Based on the preceding analysis, we propose to update the TR 26.940 as follows:

\* \* \* 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.

7.3 Complexity Analysis of an existing AI Codec: DAC

To investigate the relationship between theoretical complexity and practical performance, we conducted a detailed analysis of a publicly available AI codec, DAC (Descriptive Audio Codec) [8]. For this analysis, we used the methodology described in Clause 7.3.1 and a pretrained model from [11], which has a 44.1 kHz sample rate and an 8-kbps codec bitrate.

7.3.1 Methodology for Complexity Analysis

To provide objective and reproducible data, a standardized benchmarking methodology was employed. The methodology involves using the ONNX Runtime library [3] to execute AI models on target hardware. This approach allows for complexity performance measurement across different "execution providers." For this analysis, we used the standard CPU backend and NNAPI (Neural Networks API) [3] backend, which is Android's interface for offloading AI workloads to specialized hardware accelerators like NPUs.

For all tests, the original, unmodified pretrained model from reference [11] was used. No changes were made to the model architecture or its parameters, and no retraining was performed. The model's fully convolutional architecture allows it to process variable-length inputs; the different frame sizes tested were achieved simply by varying the length of the input audio sample provided to the model. Furthermore, no quantization (e.g., to INT8 or INT16) was applied. The original float model was used for all execution providers, including NNAPI, as quantizing this type of generative audio model can be challenging due to its sensitivity to precision.

The method is designed to measure key complexity indicators by processing audio frames of various durations. For each test configuration, warm-up iterations are performed before collecting detailed statistics over multiple runs. The primary metric gathered is the Real-Time Factor (RTF), calculated for the full end-to-end pipeline as well as for individual components (e.g., encoder and decoder stages). This allows for a comprehensive analysis of both overall complexity performance and potential bottlenecks within the model architecture.

7.3.2 Theoretical Complexity Analysis

The theoretical computational load of the DAC model was analyzed using two established profiling libraries, ptflops (v0.7.5) and thop (v2.0.17, via the ultralytics fork) [9][10], to cross-verify the results. The complexity scales with the audio frame size, increasing from approximately 1.4 GFLOP counts a 20ms frame to 31.6 GFLOP counts for a 320ms frame, as shown in Figure 1.



Figure 1: DAC Model Theoretical Complexity Comparison

The model's complexity was profiled both end-to-end and by evaluating its components separately. To simulate real-world usage, the encoder was profiled using dummy input tensors representing raw audio waveforms of shape *[1, 1, floor(sample\_rate \* duration\_in\_seconds)]*, with the duration varied to measure complexity scaling. The decoder, which takes the quantized latent representation as input, was profiled using dummy tensors of shape *[1, 1024, T]*, where *T* (time) was varied using values *[1, 3, 6, 13, 27]* to correspond to increasing audio durations.

Initial analysis (as documented in S4-251333) was conducted using older library versions (ptflops v0.7.4 and thop v1.1.1), which produced aligned results. However, when re-evaluating with the current, updated library versions cited in this document, a different set of results was obtained. While the newer libraries are also aligned with each other, their figures do not match the previous analysis. The change was traced back to a fundamental shift in the calculation methodology for *ConvTranspose1d* layers between the older and newer generations of the profiling tools.

The complexity metric we are using here (GFLOP counts per frame) represents the total number of floating-point operations required to process a single input frame of a given length. The results from both tools were highly consistent. The model has 76.9M parameters, resulting in a model size of 293 MB.

7.3.3 Real-World Inference Performance Analysis

To see how the DAC model performs in the real world, we tested it on two very different types of devices. We chose a powerful high-end desktop computer to see how fast it could run in an ideal case. We also tested it on a high-end mobile phone, which is closer to what users will actually have. This helps us understand the performance gap between a best-case scenario and a typical use case.

For our main measurement, we used the Real-Time Factor (RTF). This is an important metric because an RTF lower than 1.0 means the codec can process audio in real-time, like during a phone call. A lower RTF is always better.

7.3.4. Key Findings

Our tests gave us some important insights into how the codec performs in the real world. These findings are shown in the figures below and help explain why just looking at theoretical numbers isn't enough.

1. On the high-end desktop CPU (frequency fixed to 5.7GHz), the codec is not real-time capable with a single thread (RTF between 1.6 and 1.9). As shown in Figure 2, real-time performance (RTF between 0.67 and 0.86) is only achievable with multi-threaded execution (4 threads). However, it is still very slow for such a high-end desktop CPU.



Figure 2: DAC Model on AMD Ryzen 9 7950X Inference Performance Analysis

1. On the high-end mobile SoC (Qualcomm Snapdragon 8 Gen 2), no tested configuration achieves real-time performance. The best-case mobile RTF was 2.125 (over 2x slower than real-time), with the worst case reaching 5.884 (nearly 6x slower than real-time).
2. We had hoped that using the phone's NPU through the NNAPI backend would make the codec faster. However, the results were inconsistent (Figure 3). Sometimes it helped a little, but for one test, it actually made performance much worse than using the CPU. This shows that we cannot simply assume that using an NPU will automatically lead to better performance.



Figure 3: DAC Model on Qualcomm Snapdragon 8 Gen 2 Inference Performance Analysis

1. The most important finding is the large gap between the theoretical complexity (GFLOP counts per frame) and the actual measured performance (RTF), as shown in Figure 4. A model that seems reasonably efficient on paper (~2-5 GFLOP counts per frame) was not able to run in real-time on a top-of-the-line mobile phone. This result is the key reason why this analysis suggests that real-world testing is so important.



Figure 4: DAC Model Theoretical Complexity vs. Real-World Inference Performance

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

6.3 Design Constraint Verification

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

6.3.1 Complexity Verification

While the complexity constraints for the ULBC codec may be based on theoretical, platform-agnostic metrics (such as MACs/FLOPs for AI-based components and WMOPS for traditional signal processing components), it is essential to ensure these metrics are meaningful for real-world deployment.

Therefore, after the theoretical complexity metrics and their target values are determined, a practical verification process should be considered. To ensure the resulting codec is widely deployable, it may be beneficial for this verification process to consider the performance on a few representative devices. The approach taken by other standards bodies, such as JPEG AI, to validate feasibility across the diverse hardware ecosystem may be worth learning from.

Editor’s note: Verification process based on representative devices is FFS

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

### References

1. SP-250378 “New SID on Ultra Low Bitrate Speech Codec”.
2. ITU-T Software Tool Library 2024 User’s Manual.
3. ONNX Runtime. "NNAPI Execution Provider." ONNX Runtime Documentation. [Online]. Available: https://onnxruntime.ai/docs/execution-providers/NNAPI-ExecutionProvider.html.
4. Qualcomm, "QNN," in AI Engine Direct SDK, Jul. 07, 2025. [Online]. Available: https://docs.qualcomm.com/bundle/publicresource/topics/80-63442-50/introduction.html.
5. MediaTek Inc., " The MediaTek NPU: AI Processing Unit " [Online]. Available: https://i.mediatek.com/ai.
6. HiSilicon, "HUAWEI Da Vinci Architecture" [Online]. Available: https://www.hisilicon.com/en/products/kirin/kirin-flagship-chips/kirin-9000.
7. "Apple's Neural Engine (ANE)" Apple Wiki, Fandom. [Online]. Available: https://apple.fandom.com/wiki/Neural\_Engine.
8. R. Kumar, P. Seetharaman, A. Luebs, I. Kumar, and K. Kumar, "High-Fidelity Audio Compression with Improved RVQGAN," arXiv preprint arXiv:2306.06546, 2023.
9. V. Sovrasov, "flops-counter.pytorch." GitHub, 2019. [Online]. Available: https://github.com/sovrasov/flops-counter.pytorch.
10. Lyken, "pytorch-OpCounter." GitHub, 2018. [Online]. Available: <https://github.com/Lyken17/pytorch-OpCounter>.
11. Descript, Inc., "Descript Audio Codec (DAC) Pretrained Model," Version 0.0.1, Feb. 2023. [Online]. Available: <https://github.com/descriptinc/descript-audio-codec/releases/tag/0.0.1>
12. E. Alshina, J. Ascenso and T. Ebrahimi, "JPEG AI: The First International Standard for Image Coding Based on an End-to-End Learning-Based Approach," in IEEE MultiMedia, vol. 31, no. 4, pp. 60-69, Oct.-Dec. 2024, doi: 10.1109/MMUL.2024.3485255.