March 31, 2025

Qualcomm

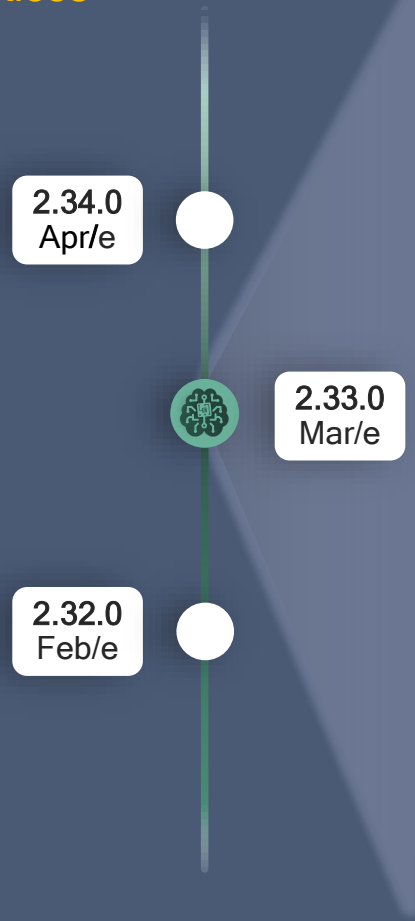
Qualcomm®
AI Runtime SDK
2.33.0 – Release Notes

Qualcomm Technologies, Inc.

QAIRT SDK

Partner Release Notes (NDA)

Notices



Important notes:

- **Transition to “Qualcomm® AI Runtime SDK” Product in QPM**
  - What’s Changing?
    - Starting with SDK v2.32.0, QAIRT package will now be available under "Qualcomm® AI Runtime SDK" in QPM, supporting all SoCs.
    - Separate add-ons will be available, controlled by customer entitlements. Clients with the proper entitlement will automatically install both the main package and the corresponding add-ons.
  - Additional Details
    - The QAIRT SDK provides binaries for Linux/X64, Windows/X64, Android, WoS, and LE/Yocto.
    - The QAIRT product was previously associated with the names:
      - Qualcomm® AI Engine Direct
      - Qualcomm® Neural Processing SDK
    - The older product names will remain available in QPM for accessing previous releases, i.e., SDK versions before v2.32.0. This change will also apply to future monthly releases.

QAIRT SDK

Notices (cont.)

New features

Deprecations

Removals

Dependencies

Compatibility

Etc.

- **System Level Changes** - Starting in SDK 2.22.0
  - Ubuntu 22.04
  - Python 3.10 (Default Python on Ubuntu 22.04)
    - Replaces Python 3.8 Support
  - Clang 14 on X86/Linux (Default Clang on Ubuntu 22.04)
  - Android NDK to r26c
  - Training Frameworks
    - Retire TF 1.15 Support (will also remove Python 3.6 support)

- **ML-Framework version upgrades**
  - ONNX Conversion now supports ONNX 1.16 (opset 21) starting from SDK 2.25.0. Please refer to SDK documentation for Operator support details.

- **Officially supported SDK host is Linux Ubuntu 22.04 (Jammy)**
  - Use of Python 3.10 and libc++14 is required (versions SDK has been tested with)
  - Note that Python 3.8 is no longer supported

- **Secure Process Domain (PD) support**
  - Enables use of QNN HTP/DSP backends in SecurePD applications
  - Support provided with add-on SDK package: SecurePD
    - See documentation in add-on SDK for integration details
  - SoC support is limited to:
    - Android: Snapdragon (SD): 8Gen3, 8Gen2 (HTP), QCS610 (DSP)

QAIRT SDK

Notices (cont.)

New features

Deprecations

Removals

Dependencies

Compatibility

Etc.

- **New features in this release**
  - See “New features added, and key issues resolved” slide
- **Deprecations in this release**
  - None
- **Removal of deprecated features in this release**
  - None
- **Planned removal of deprecated features**
  - We are retiring support for targets based on aarch64-ubuntu-gcc7.5 from this release (v2.27.0) onwards. Please note that the QRB5165.UBUN.1.0 SP, which was the primary target using this toolchain, has already reached its end-of-life.
  - Starting from the 2.35 release of the SDK, the fp16_relaxed_precision used in QNN SDK tools and QNN_HTP_GRAPH_CONFIG_OPTION_PRECISION in QNN HTP backend's QnnHtpGraph_CustomConfig_t, will no longer be effective. The HTP backend will always execute floating point ops using float16 math on SoCs with float support, equivalent to setting the value of fp16_relaxed_precision to 1.
- **HTP User Process Domain (PD) support**
  - QNN HTP backend by default load into **UnsignedPD**
  - Only unsigned Skel libraries are included in SDK

Deprecated = still supported (bug fixes only) but subject to removal in future release. Use not recommended.
API and features that have been deprecated are marked as such in API header files and/or SDK documentation.

New features added and key issues resolved

Selected changes in this release

- Refer to QNN_ReleaseNotes.txt & Releasenotes.txt (for SNPE) in the QAIRT SDK root folder for complete details on new features, bug fixes, and known issues.
- This slide summarizes additional changes not captured in the SDK release notes.

ID	Key Features Added
F1	
F2	

ID	Known Issues
K1	HTP: Potential Performance Changes in Specific Models. Starting from v2.30.0, we included key updates to our HTP backend, which generally improved performance. However, some models may still experience changes in performance characteristics. We have made several improvements in this release, but some issues remain. We are actively working to fine-tune this update to ensure consistent performance across all models and will continue to roll out improvements in future releases.
K2	HTA backend for QRB5165.LE based on aarch64-oe-linux-gcc11.2 platform is not supported
K3	SNPE: snpe-net-run --enable_init_cache option OR API SNPEBuilder::setInitCacheMode() / Snpe_SNPEBuilder_SetInitCacheMode() are currently non-functional in DSP (v66) and AIP backends; fix planned for next release. {131929}
K4	Op:HTP: The HTP backend may incorrectly report support for updatable tensors on certain Ops, leading to failures during context binary generation. This will be addressed in the next release. {129714}
K5	HTP: Graph execution errors may occur due to ION memory deregistration failures. {129731}
K6	HTP: LoRAv2 models may fail with a memRegister error. Issue under investigation {126288}
K7	SDK: QNN SampleApp may fail to build. Will be addressed in next release {131442}

Gen AI feature Compatibility

Intended support is on SOCs with v79, v75 and v73

Feature/Chipset	SM8750	SM8650	SM8550	SM8635	SM7550	SM7635	SC8380X
Init Cancellation	☑	☑	☑	☑	☑	☑	☑
Lora v2	LVM: ☑ LLM: ☑	LVM: ☑ LLM: ☑	LVM: ☑ LLM: ☑	LVM: ☑* LLM: ☑	LVM: ☑* LLM: ☑	LVM: ☑* LLM: ☑	☑K2
Speculative Decoding	☑	☑	☑ K1	☑	☑	☑	☑
Self-Speculative Decoding	☑	☑	☑	☑	☑	☑	☑
Lookahead Decoding	☑	☑	☑	☑	☑	☑	☑
Low Power Block Quantization	☑	☑	☑	☑	☑	☑	☑
Models: LLaMA v3 8B	☑	☑	☑	☑	☑	☑	☑
Models: SDXL	☑	☑	☑	☑	☑	☑	☑

Note:
- * If LVM use case requires FP16, please note that it cannot be supported on these SoCs, as they do not include FP16 support
- All the above use cases are tested with Llama v2 7b as base model

ID	Gen AI Known Issues
K1	Observing out of memory issue in resulting in application being killed while executing SPD SM8550 {119663}
K2	LoRA v2 feature on SC8380X is validated with context binaries generated with x86 ubuntu host - {116561}

Mobile

Devices Used for Testing			
SOC	**Device**	**OS Version**	**Software Version**
SM8750	QRD SM8750	Android 14	Pakala.LA.1.0.r1-01061-PERF.INT-1
SM8650	QRD SM8650	Android 14	Lanai.LA.2.0-00070-GKI.INT-1
SM8635	QRD SM8635	Android 14	Palawan.LA.2.0-00078-GKI.INT-1
SM8550	QRD SM8550	Android 14	Kailua.LA.2.0.1.r1-00020-GKI.INT-1
SM8450	QRD SM8450	Android 14	Waipio.LA.2.0.2.r1-00007-GKI.INT-1
SM8350	QRD SM8350	Android 14	Lahaina.LA.2.0.3-00014-STD.INT.QGKI-2
SM8250	QRD SM8250	Android 13	SM8250_SDX55.LA_TN.2-0-2_2-5-1-00019
SM7675	QRD SM7675	Android 14	Palawan.LA.2.0-00078-GKI.INT-1
SM7450	QRD SM7450	Android 14	Fillmore.LA.2.1.2.r1-00029-GKI.INT-1
SM7325	MTP SM7325	Android 14	Kodiak.LA.2.0.3-00014-STD.INT.QGKI-1
SM7250	MTP SM7250	Android 13	Saipan.LA.3.2.1-00014-STD.INT-1
SM7225	QRD 7225	Android 13	Bitra.LA.3.2.1-00014-STD.INT-1
SM6375	MTP SM6375	Android 14	Strait.LA.2.1.2-00011-STD.INT-1
SM6350	QRD SM6350	Android 13	Bitra.LA.3.2.1-00014-STD.INT-1
SM6225	QRD SM6225	Android 14	Divar.LA.3.0.1-00012-GKI.INT-1
SM6115	QRD SM6115	Android 12	Kamorta.LA.2.0-00118-STD.INT-1
SM4350	MTP SM4350	Android 14	Mannar.LA.2.5.1-00023-GKI.INT-1
SM4250	MTP SM4250	Android 11	Kamorta.LA.2.0-00118-STD.INT-1

IOT

Devices Used for Testing			
SOC	**Device**	**OS Version**	**Software Version**
QRB5165 LE	MTP QRB5165	OpenEmbedded	QRB5165.LE.1.0-00010-STD_FULLSTACK_LP5.INT-1
QCS610 LE	MTP QCS610	OpenEmbedded	QCS610.LE.1.0-00094-STD.INT.64bit-1
QCS610 LA	MTP QCS610	Android 10	QCS610.LA.1.0-00026-SECBOOT.INT-1
QCM6490	MTP QCM6490	Android 12	QCM6490.LA.2.0-00030-STD.INT.MSL-1
QCM6125	MTP QCM6125	Android 11	QCM6125.LA.2.0-00011-STD.INT-1
QCS4290	MTP QCM6125	Android 11	QCM4290.LA.1.0.1-00047-STD.APQ.INT-1
QRB5165 UBUN	MTP QRB5165	Ubuntu	QRB5165.UBUN.2.0-00275-POP_LP5.INT-1
QCS8550 LA	MTP QCS8550	Android 13	QCM8550.LA.1.1-00031-GKI.INT-1
QCS9100 LE	QCS9100	OpenEmbedded	QCS9100.LE.1.0.r1-00193-STD.INT-1

Devices used for Testing

XR

Devices Used for Testing			
SOC	Device	OS Version	Software Version
SXR2250P (XR Gen2)	MTP SXR2250P	Android 14	Halliday.LA.2.0-00046-PERF.INT-1
SXR2230P (XR Gen2)	MTP SXR2230P	Android 12	Halliday.LA.1.1-00147-PERF.INT-1
Luna	MTP Luna	v73	Yes
SXR1230P_LE	MTP SXR1230P_LE	v73	Yes

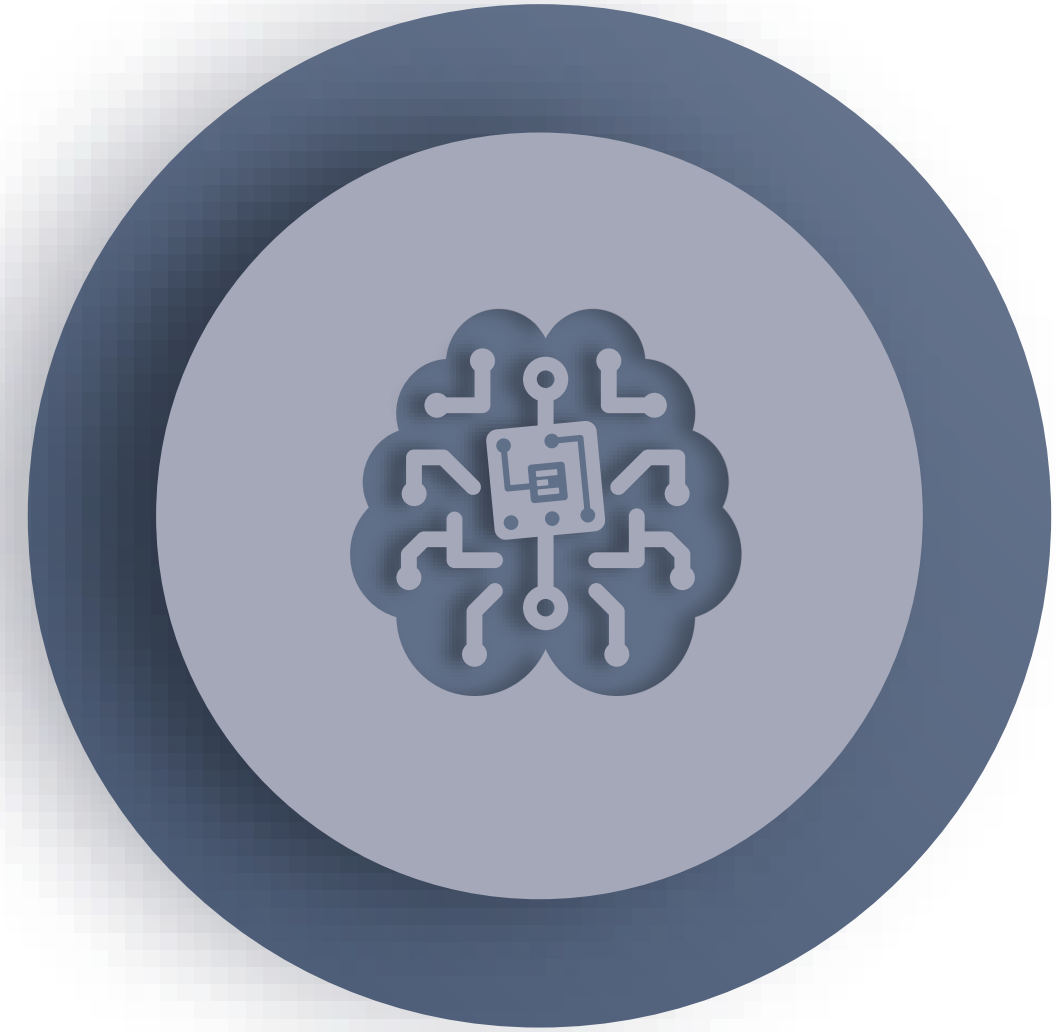
Compute

Devices Used for Testing			
SOC	Device	OS Version	Software Version
SC8380X	Hamoa	Win 11	APSS.WP_HA.1.0-07400-SC8380XRELSFNWZA-4
SC8180X	PRIMUS	Win 10	APSS.WD_PO.1.1-17800-SC8180XRELSFNWZA-2
SC8280X	MAKENA	Win 11	APSS.WP_MK.1.0-13200-SC8280XRELSFNWZA-3
SC7280X	KODIAK	Win 10	APSS.WP_KD.1.0-04000-SC7280RELSFNWZA-2

Devices used for Testing

Release Metrics

Performance and accuracy for this release.



Inference speed (inf/s) by network, chipset and core (clang 64-bit arch)

SoC	INCEPTION_V3_2016					RESNET50					VGG16					YOLOV5					MOBILENET_V3_MINIMAL				
	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8
8 Gen 4 (8750)	2173.9	113.8	60.4	28	62	2212.4	148.1	94.2	37.8	67.2	453.1	37.4	26.4	9.9	27.3	1187.6	72.5	46.8	16.1	23.6	8000	500.8	449.4	415.6	415.1
8 Gen 3 (8650)	1834.9	83.3	44.1	12.4	23.8	1960.8	101.3	66.2	17.4	30.7	391.1	36.7	22.7	4.8	11.1	833.3	67.9	44.2	7.7	9.8	6802.7	375.8	331.5	161.3	193.2
8s Gen 3 (8635)	977.5	69.7	33.9	11	22.4	1340.5	85.6	50.7	15	29.3	286.6	30	14.6	4.1	10.7	532.8	48	29.5	7	9.3	4854.4	407.2	359.2	138.5	185.8
8 Gen 2 (8550)	1388.9	69.6	34.5	9.8	24.4	1485.9	84.9	50.1	14.9	29.4	303.7	30.5	17.7	4.5	11.3	613.5	55.8	34.7	7.1	9.9	5208.3	327.1	301.9	131.3	200.9
8+ Gen 1 (8475)	1102.5	67.2	36	11.4	24	1243.8	86.4	51.1	14.9	29.8	255.7	29.1	15.7	4.2	11.1	457.7	47.2	30.5	6.6	9.8	4081.6	388.2	342.2	141.8	207.6
8 Gen 1 (8450)	1107.4	63.3	33.5	10.9	24.2	1242.2	80.8	47.8	13.8	30.1	256.3	27.3	14.5	4.2	11	473.7	44.4	28.3	6.9	10	4184.1	348.6	297.2	141.7	214.5
888+ (8350P)	579.7	42	19.4	10.5	30.5	730.5	47.8	25.8	13.3	38.5	180.9	21	9.2	3.6	13.8	306.7	29.3	16.4	6.8	12.5	2267.6	207.5	205.1	140.2	216.1
888 (8350)	566.3	45.3	20	9.5	25.8	765.1	52.8	27.2	12.6	34	186.5	22.6	9.3	3.7	11.9	321.4	32.3	17.1	6.5	10.3	2314.8	262.7	254.6	131.6	213.7
865 (8250)	116.8	31.6	14.4	6.7	17.8	115.2	37.4	19.7	8.4	22.9	44.2			2.2	7.5	38.1	21.6	12	4.4	6.9	336.7	212.2	190.2	102.4	132.7
7+ Gen 3 (7675)	944.8	46.7	23.3	12.6	1307.2	1297	57.3	35.26	15.8	36.67	281.1	22	11.7	4.4	13.3	520.6	33	22	7.5	12.2	4878	304.6	272.6	192.5	312.79
7+ Gen 2 (7475)	871.8	47.3	24.9	10.6	23.1	1088.1	64.1	36	14.2	29.3	223.8	20.8	10.7	4	10.6	416.1	33.9	20.8	6.5	9.5	3703.7	274.7	228	139.8	202.8
7 Gen 1 (7450)	337.8	23.1	11.4	8.6	19	418.9	25.4	15.2	10.3	23.7				3.1	8.5	173.9	14.1	8.7	5.2	7.5	1851.9	139.5	130.7	116.6	148.5

NOTES:

- Performance is calculated according to the methodology described in the “Performance Measurement Methodology” Application Note.
- Using the **BURST** mode will produce larger numbers, see the “Burst mode Performance” Application Note..

Runtime	Data format	Input format
AIP	8 bit quantized	TF8 user buffers
DSP	8 bit quantized	TF8 user buffers
G16	16-float data and math	32-float
GPU	16-float data, 32 math	32-float
CPU	32-float data and math	32-float

Configurations of the reported runtimes.

Inference speed (inf/s) by network, chipset and core (clang 64-bit arch)

SoC	INCEPTION_V3_2016					RESNET50					VGG16					YOLOV5					MOBILENET_V3_MINIMAL				
	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8
778G (7325)	297.2	21	11.6	7.5	18.9	373.3	24.8	15.2	9.7	23.5				2.5	10.3	160.72	13.39	8.55	5.53	8.88	1264.2	160.1	147	110.5	143.8
765 (7250)	65.7	10.9	7	4	10.3	65.6	12.4	9.6	5.3	14.3	24.8			1.5	4.7	23.7	6.8	5.1	3.3	5.1	242.5	118.6	110.1	59.9	83.2
695 (6375)	46	11	6.8	5.3	13.3	46.8	12.3	9.3	6.8	16.3	17.6			1.9	6.1	17.4	6.4	4.8	3.4	5.7	189.7	116.7	99.6	73.8	109.9
690 (6350)	39.4	8	4.9	4.6	12.3	39.7	9.3	6.9	5.3	15.6	15.1			1.5	5.2	14.1	5.1	3.6	3.1	5.2	156.1	95.1	84.7	64.2	101.1
680 (6225)	52.8	3	2.2	4.9	7.7	53.1	3.6	2.8	5.9	9.9	20.9			1.7	3.4	18.5	2.3	1.7	2.8	3.7	202.5	47.7	44.8	63.3	83.4
4 Gen 2 (4450)		3.9	2.5	5.2	14.4		4.5	3.1	6.2	17				1.8	6.5		2.4	1.8	3.1	6.5		52.2	48.6	62.5	111.2
480 (4350)	54	10	6.9	5.4	12.1	53	12.2	9.4	6.3	15	19.6			1.8	5	18	6.1	4.1	3.5	4.9	201.5	102.1	101.7	63	107.4
460 (4250)	27.1	2	1.3	3.1	4	27.5	2.2	1.7	4.1	5.4	11.5			1.2	1.8	10	1.2	0.9	1.9	1.8	102.9	32	27.1	37.2	27.4

NOTES:

- Performance is calculated according to the methodology described in the “Performance Measurement Methodology” Application Note.
- Using the **BURST** mode will produce larger numbers, see the “Burst mode Performance” Application Note..

Runtime	Data format	Input format
AIP	8 bit quantized	TF8 user buffers
DSP	8 bit quantized	TF8 user buffers
G16	16-float data and math	32-float
GPU	16-float data, 32 math	32-float
CPU	32-float data and math	32-float

Configurations of the reported runtimes.

Inference speed (inf/s) by network, on AIP targets

SoC	INCEPTION_V3_2016					RESNET50					VGG16					YOLOV5					MOBILENET_V3_MINIMAL				
	AIP	DSP	G16	GPU	CPU	AIP	DSP	G16	GPU	CPU	AIP	DSP	G16	GPU	CPU	AIP	DSP	G16	GPU	CPU	AIP	DSP	G16	GPU	CPU
865 (8250)	116.8	116.8	31.6	14.4	6.7	66.1	115.2	37.4	19.7	8.4	65.7	44.2			2.2		38.1	21.6	12	4.4	314.1	336.7	212.2	190.2	102.4
QRB5165	86.6	71.7	29	13.8	5.7	36.6	73.8	34	19	9	58.9	34.4			3.3		29	20.6	11.4	4	230.4	239.6	166.8	151.1	64.2
QRB5165_LE		118.6	30	15	5.2		117.5	31.7	20.7	8.1		44.8			3		38.2	19	12.4	3.3		348.9	211.6	195.8	59.2
765 (7250)	49.9	65.7	10.9	7	4	15.9	65.6	12.4	9.6	5.3		24.8			1.5		23.7	6.8	5.1	3.3	96.1	242.5	118.6	110.1	59.9
690 (6350)	36.8	39.4	8	4.9	4.6	14.5	39.7	9.3	6.9	5.3	22.9	15.1			1.5		14.1	5.1	3.6	3.1	85.4	155.2	95.1	84.7	66.1

NOTES:
- See the previous slide for per-core (runtime) configuration.
- Performance is calculated according to the methodology described in the “Performance Measurement” Application Note.

Inference speed (inf/s) by network, chipset and core (Android clang 64-bit arch)

SOC	INCEPTION_v3_2016						RESNET50_V1						VGG_16						YOLO_V5						CNNS_MOBILENET_V3_MINIMAL					
	HTP	HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU	HTP	HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU	HTP	HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU	HTP	HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU	HTP	HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU
8 Elite (8750)	2096.4	809.72	35.90	116.47	59.02	25.93	2155.1	931.10	53.27	154.23	65.17	35.12	451.88	190.62	18.76	38.76	26.94	9.33	1090.5	434.22	32.98	78.40	23.37	15.35	6535.9	3831.4	318.88	542.30	352.24	335.23
8 Gen 3 (8650)	1795.3	564.33	28.60	79.96	23.94	12.31	1865.6	717.88	44.82	101.05	27.73	17.24	390.78	149.72	14.87	36.56	11.01	4.73	813.67	290.78	18.43	33.04	10.15	7.57	5780.3	3095.9	238.15	372.02	190.84	145.77
8s Gen 3 (SM8635)	962.46	-	20.78	67.04	22.05	11.09	1312.3	-	30.96	84.82	27.83	14.89	283.21	-	8.85	29.83	10.55	4.04	504.03	-	12.56	27.50	9.62	6.95	4464.2	-	237.76	362.06	183.82	129.27
8 Gen 2 (8550)	1345.9	504.29	22.57	67.10	23.01	11.58	1455.6	593.82	36.21	83.65	27.40	14.34	301.93	127.62	12.00	30.33	11.94	4.20	611.25	233.21	16.77	31.34	10.59	6.80	4716.9	2673.8	213.86	308.17	206.65	116.90
8+ Gen 1 (8475)	1077.5	306.47	24.90	65.31	23.82	10.80	1196.1	419.46	35.27	84.88	27.91	14.15	254.07	88.10	10.38	29.02	11.05	4.15	451.47	175.87	16.82	32.36	10.05	6.51	3802.2	1848.4	215.84	359.20	205.17	125.49
8 Gen 1 (8450)	1088.1	307.69	23.16	61.44	24.25	10.57	1210.6	421.05	32.95	80.37	27.80	13.26	255.23	82.84	9.65	27.09	10.88	4.16	470.15	178.67	15.65	30.88	10.18	6.62	3831.4	1879.7	201.90	326.48	207.38	115.50
888+	564.65	-	15.18	42.82	-	10.30	698.81	-	19.91	49.39	-	13.18	178.06	-	7.30	21.78	-	3.59	307.50	-	10.63	21.34	-	6.42	2132.2	-	157.63	212.22	-	125.33
888 (8350)	607.90	-	15.30	45.78	25.69	9.46	778.82	-	19.90	51.53	32.44	12.49	185.87	-	7.42	22.48	11.94	3.66	321.75	-	11.02	22.89	10.61	6.39	2092.0	-	180.83	259.34	205.13	124.32
7+ Gen 3 (SM7675)	918.27	-	15.68	49.86	30.21	12.37	1257.8	-	23.83	63.69	36.18	16.23	278.63	-	7.70	22.62	12.35	4.21	509.16	-	10.77	23.32	11.94	6.54	3831.4	-	182.72	277.01	287.44	110.62
7s Gen 3 (SM7635)	263.99	-	5.01	18.09	25.19	10.60	412.54	-	5.72	17.42	30.01	13.54	-	-	1.32	10.22	11.04	3.44	167.14	-	2.33	6.43	9.68	6.77	1865.6	-	86.19	117.55	180.54	108.75
7 Gen 1 7450	337.84	95.88	8.61	22.84	19.30	8.36	418.94	128.82	11.05	25.33	23.31	9.92	-	-	-	-	8.46	3.05	173.79	58.93	6.07	11.80	7.76	5.05	1838.2	751.31	90.29	124.63	148.52	100.41
7+ Gen 2 7475	864.30	-	17.17	46.42	22.85	10.31	1063.8	-	24.92	63.74	27.51	13.45	221.78	-	7.25	20.74	10.59	3.98	412.37	-	11.92	25.48	9.80	6.33	3460.2	-	163.69	256.08	200.04	124.69
778G (7325)	312.79	-	8.70	20.80	-	7.93	391.70	-	11.13	24.66	-	9.47	-	-	-	-	-	2.52	160.57	-	5.97	12.00	-	4.94	1522.0	-	96.37	163.08	-	100.84

Runtime	Data format	Input format
HTP/DSP	8 bit quantized	8-bit
HTP_FP16	16-float data and math	**16-float**
GPUF32	32-float data and math	32-float
GPUF16	16-float data and math	16-float
CPU	32-float data and math	32-float
CPU_INT8	8 bit quantized	**8-bit**
HTA	8 bit quantized	8-bit

NOTES:

- Performance is calculated according to the methodology described in the Release documentation “Benchmarking” section
- HTP/DSP performance numbers are measured in unsignedPD
- Above numbers collected using the BURST mode for HTP and DSP

Configurations of the reported runtimes.

Inference speed (inf/s) by network, chipset and core (Android clang 64-bit arch)

SOC	INCEPTION_v3_2016					RESNET50_V1					VGG_16					YOLO_V5					CNNS_MOBILENET_V3_MINIMAL				
	DSP	HTA	GPU F32	GPU F16	CPU	DSP	HTA	GPU F32	GPU F16	CPU	DSP	HTA	GPU F32	GPU F16	CPU	DSP	HTA	GPU F32	GPU F16	CPU	DSP	HTA	GPU F32	GPU F16	CPU
865 (8250)	113.37	110.07	11.11	31.06	6.57	111.53	-	15.12	36.89	8.34	43.71	64.17	-	-	2.15	36.92	-	8.06	16.36	4.39	347.71	290.02	134.37	207.08	99.76
695 (6375)	45.99	-	5.76	10.62	5.18	45.63	-	8.07	12.12	6.58	17.57	-	-	-	1.87	16.41	-	3.77	5.75	3.30	179.50	-	76.06	104.95	68.20
765 (7250)	62.72	71.01	5.44	10.67	3.93	61.74	-	7.31	12.03	5.76	24.14	38.90	-	-	1.59	22.23	-	3.61	5.96	2.47	249.81	183.55	78.76	105.14	57.27
690 (6350)	38.08	47.72	4.24	8.00	4.51	38.21	-	5.93	9.17	5.25	14.94	25.77	-	-	1.46	13.43	-	2.90	4.61	3.01	156.86	143.51	60.86	83.95	66.02
680 (6225)	50.13	-	1.84	3.00	4.65	50.90	-	2.27	3.66	5.92	20.75	-	-	-	1.63	17.09	-	1.48	2.19	2.92	207.68	-	35.88	46.14	62.15

NOTES:
- Performance is calculated according to the methodology described in the Release documentation “Benchmarking” section
- HTP/DSP performance numbers are measured in unsignedPD
- Above numbers collected using the BURST mode for HTP and DSP

Runtime	Data format	Input format
HTP/DSP	8 bit quantized	8-bit
HTP_FP16	16-float data and math	**16-float**
GPUF32	32-float data and math	32-float
GPUF16	16-float data and math	16-float
CPU	32-float data and math	32-float
CPU_INT8	8 bit quantized	**8-bit**
HTA	8 bit quantized	8-bit

Configurations of the reported runtimes.

Inference speed (inf/s) by network, chipset and core

SoC()	INCEPTION_V3_2016					RESNET50					VGG16					YOLOV5					MOBILENET_V3_MINIMAL				
	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8	DSP	G16	GPU	CPU	CPU_INT8
QRB5165	71.7	29	13.8	5.7	23.7	73.8	34	19	9	31.1	34.4			3.3	9.3	29	20.6	11.4	4	8.7	239.6	166.8	151.1	64.2	170.8
QRB5165_LE	118.6	30	15	5.2	25.9	117.5	31.7	20.7	8.1	31.1	44.8			3	10	38.2	19	12.4	3.3	10	348.9	211.6	195.8	59.2	186.1
QCS610_LA	45.7	3.4	1.9	3.9		45.2	3.5	2.5	5.3		17.7			1.8		16.3	2.1	1.4	2.3		156.4	47	39.3	39	
QCS4290	43.5	2.7	1.8	3.5		42.1	3	2.3	4.8		17.6			1.4		15.3	1.8	1.3	2.3		149.8	42	35.7	38.3	
QCM6490	382.1	26.9	15.9	9.1	24.9	483.1	32	20.1	10.6	31.9	0.2			2.9	11.1	175.6	17.6	11.6	5.9	10.1	1457.7	202.4	179.5	117.1	202.7
QCM4490		4.4	2.6	6.9	17.6		5.1		8.6														53.5	97.8	154.1
QCS8550_LA	1414.4	59.1	30.8	16.6	40.6	1506	69.3	44.2	22.2	45.4	305.2	27.6	16.2	5.2	17	642.3	48.3	31.1	9.3	16	5263.2	294.1	274.3	217.1	323
QCS9100_LE	1169.5			11.8	25.2	1171			17	34.2	307.3			6.7	10.2	543.8			6.1	9.8	3236.2			94	151
AR2 Gen1 SAR2130P	166.3			1.6		195.6			2.1							105.8			1		752.4			15.6	
XR2 Gen2 (SXR2250P)	1131.2	70.6	36.9	13.4	26.1	1065	82.4	54.8	16.3	34	233.8	30.8	18.7	3.9	12	437.6	52.7	36.5	8.5	9.9	3891.1	349.2	320.6	195.2	188.4

NOTES:

- Performance is calculated according to the methodology described in the “Performance Measurement Methodology” Application Note.
- Using the **BURST** mode will produce larger numbers, see the “Burst mode Performance” Application Note..

Runtime	Data format	Input format
AIP	8 bit quantized	TF8 user buffers
DSP	8 bit quantized	TF8 user buffers
G16	16-float data and math	32-float
GPU	16-float data, 32 math	32-float
CPU	32-float data and math	32-float

Configurations of the reported runtimes.

Inference speed (inf/s) by network, chipset and core (Android clang 64-bit arch)

SOC	INCEPTION_v3_2016						RESNET50_V1						VGG_16						YOLO_V5						CNNS_MOBILENET_V3_MINIMAL					
	HTP/DSP	HTA/HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU	HTP/DSP	HTA/HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU	HTP/DSP	HTA/HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU	HTP/DSP	HTA/HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU	HTP/DSP	HTA/HTP_FP16	GPU F32	GPU F16	CPU_INT8	CPU
QRB5165U	105.24	80.02	10.78	28.49	-	5.54	105.71	-	14.82	34.97	-	8.55	42.17	57.19	-	-	-	3.19	34.75	-	7.65	14.61	-	3.71	300.57	239.46	118.01	165.95	-	59.79
QRB5165_LE	116.56	-	11.07	28.98	-	5.09	114.21	-	15.00	31.52	-	7.73	44.69	-	-	-	-	2.75	36.45	-	8.01	15.06	-	3.15	355.24	-	137.08	209.12	-	51.68
QCS610LE	45.31	-	1.63	3.23	-	3.38	44.56	-	2.00	3.49	-	5.03	-	-	-	-	-	1.52	15.31	-	1.24	2.04	-	1.79	154.08	-	30.95	43.05	-	35.68
QCM6490	173.10	-	5.24	12.74	-	6.27	262.47	-	6.84	15.30	-	7.63	-	-	-	-	-	2.37	91.56	-	3.64	8.04	-	3.87	1076.4	-	62.10	100.19	-	76.72
SM4490	-	-	2.20	4.24	16.04	6.75	-	-	3.12	4.92	20.11	8.41	-	-	-	-	7.52	2.23	-	-	1.66	2.75	7.09	4.41	-	-	44.57	57.43	150.44	86.50
SM4450	-	-	2.21	4.25	16.47	6.88	-	-	3.14	5.11	20.98	8.47	-	-	-	-	7.75	2.49	-	-	1.56	2.65	7.07	4.44	-	-	44.67	57.36	150.47	87.05
AR2 Gen1 SAR2130P	164.39	-	-	-	-	1.62	180.21	-	-	-	-	2.08	-	-	-	-	-	-	93.08	-	-	-	-	1.02	779.42	-	-	-	-	15.39
AR1 Gen1 SAR1130P	174.64	-	6.57	12.04	-	3.38	190.55	-	8.21	14.84	-	4.40	-	-	-	-	-	1.30	-	-	-	-	-	-	1052.6	-	77.86	103.43	-	36.67
XR2 Gen2 (SXR2250P)	1109.8	289.77	23.79	68.02	-	12.97	1168.2	398.72	36.57	80.13	-	12.72	250.38	75.21	12.84	30.29	-	4.06	429.37	161.71	16.08	28.54	-	6.85	3597.1	1763.6	229.46	336.25	-	157.53
QCS8550	1355.0	507.87	20.17	56.82	40.15	15.40	1453.4	597.73	32.40	71.02	44.34	20.30	303.21	127.80	11.15	27.28	16.26	4.82	636.94	241.90	14.02	24.59	15.82	9.07	4545.4	2512.5	168.98	239.52	307.50	117.00
QCS9100LE	1020.4	288.27	-	-	24.70	12.17	1243.7	406.17	-	-	29.89	15.72	279.56	89.21	-	-	9.83	6.56	567.54	173.91	-	-	8.52	5.70	3289.4	1773.0	-	-	176.27	63.04

NOTES:

- Performance is calculated according to the methodology described in the Release documentation “Benchmarking” section
- HTP/DSP performance numbers are normally measured in unsignedPD. Some are measured in SignedPD due to platform support limitation: 690 DSP
- Above numbers collected using the BURST mode for HTP and DSP

Runtime	Data format	Input format
HTP/DSP	8 bit quantized	8-bit
HTP_FP16	16-float data and math	**16-float**
GPUF32	32-float data and math	32-float
GPUF16	16-float data and math	16-float
CPU	32-float data and math	32-float
CPU_INT8	8 bit quantized	**8-bit**
HTA	8 bit quantized	8-bit

Configurations of the reported runtimes.

Inference speed (inf/s) by network, chipset and core

SoC()	INCEPTION_V3_2016			RESNET50			VGG16			YOLOV5			MOBILENET_V3_MINIMAL		
	DSP	G16	GPU	DSP	G16	GPU	DSP	G16	GPU	DSP	G16	GPU	DSP	G16	GPU
SC8380X	1036	110	55	1160	124	83	308	44	29	542	88	59	1818	547	502
SC8280X	575	-	-	703	-	-	184	-	-	296	-	-	1468	-	-
SC8180X	105	-	-	105	-	-	41	-	-	37	-	-	290	-	-
SC7280X	322	-	-	347	-	-	-	-	-	161	-	-	929	-	-

NOTES:

- Performance is calculated according to the methodology described in the “Performance Measurement Methodology” Application Note.
- Using the **BURST** mode will produce larger numbers, see the “Burst mode Performance” Application Note..

Runtime	Data format	Input format
AIP	8 bit quantized	TF8 user buffers
DSP	8 bit quantized	TF8 user buffers
G16	16-float data and math	32-float
GPU	16-float data, 32 math	32-float
CPU	32-float data and math	32-float

Configurations of the reported runtimes.

Inference speed (inf/s) by network, chipset and core (VC22-64-bit aarch)

SOC	INCEPTION_v3_2016			RESNET50_V1			VGG_16			YOLO_V5			CNNS_MOBILENET_V3_MINIMAL		
	HTP/DSP	GPU F32	GPU F16	HTP/DSP	GPU F32	GPU F16	HTP/DSP	GPU F32	GPU F16	HTP/DSP	GPU F32	GPU F16	HTP/DSP	GPU F32	GPU F16
SC8380X	1196	37	108	1300	59	124	333	20	44	513	31	65	2994	357	569
SC8280X	621	-	-	798	-	-	186	-	-	287	-	-	2044	-	-
SC7280X	340	-	-	383	-	-	-	-	-	167	-	-	1182	-	-
SC8180X	108	-	-	106	-	-	42	-	-	34	-	-	324	-	-

Runtime	Data format	Input format
HTP/DSP	8 bit quantized	8-bit
HTP_FP16	16-float data and math	**16-float**
GPUF32	32-float data and math	32-float
GPUF16	16-float data and math	16-float
CPU	32-float data and math	32-float
CPU_INT8	8 bit quantized	**8-bit**
HTA	8 bit quantized	8-bit

NOTES:

- Performance is calculated according to the methodology described in the Release documentation “Benchmarking” section
- Above numbers collected using the BURST mode for HTP and DSP

Configurations of the reported runtimes.

Inference accuracy per metric, by network and runtime(clang 32-bit arch)

Runtime	INCEPTION_V3 2016_OPT			MOBILENET_V1 QUANT_AWARE		
	mAP	Top-1	Top-5	mAP	Top-1	Top-5
Tensorflow or Caffe	81.74	76.79	93.33	67.32	60.15	83.06
SNPE CPU	81.85	76.86	93.47	67.32	60.15	83.06
SNPE AIP	82.04	77.08	93.54	75.57	70.05	89.15
SNPE DSP (V66)	81.02	76.2	92.63	75.48	70	88.8
SNPE DSP (V68)				75.34	70.17	89.01
SNPE DSP (V69)				75.34	70.17	89.01
SNPE DSP (V73)				75.34	70.17	89.01
SNPE DSP (V75)				75.34	70.17	89.01
SNPE DSP (V79)				75.34	70.17	89.01
SNPE GPU	81.51	76.58	93.17	75.70	70.36	89.32
SNPE GPU16	81.49	76.62	93.18	75.71	70.38	89.32

NOTES:

- Top-1 and Top-5 represent the accuracy percentage on the most likely and 5-most-likely predicted classes, on average
- For the mAP definition, and for an explanation of the measurement methodology see the “Precision Measurement” application note
- Image classification validation: ImageNet 2012 (ILSVRC2015/CLS_LOC/Data/CLS-LOC/val)
- Values are measured on a variety of devices and are consistent across devices.
- DSP (V66/V68/V69/V73/V75/V79) numbers are calculated using quantized DLCs with TF8 user buffers input format.

Inference accuracy per metric, by network and runtime (Android clang 64-bit arch)

Runtime	INCEPTION RESNET_V2			INCEPTION_V3_2016			MOBILENET_V1_QUANT_AWARE		
	mAP	Top-1	Top-5	mAP	Top-1	Top-5	mAP	Top-1	Top-5
Tensorflow (5K)	84.96	78.56	94.18	83.06	76	92.6	77.5	70.06	89.3
QNN CPU-FP32 (5K)	84.96	78.56	94.16	83.06	76	94.18	77.36	69.98	89.22
QNN HTP-FP16 (V79)(5K)	84.96	78.6	94.2	83.09	76.0	92.58	77.46	69.98	89.22
QNN HTP-FP16 (V75)(5K)	84.96	78.6	94.2	83.09	76.0	92.58	77.46	69.98	89.22
QNN HTP-FP16 (V73)(5K)	84.96	78.58	94.2	83.09	76.02	92.58	77.46	70.02	89.26
QNN HTP-FP16 (V69)(5K)	84.96	78.6	94.2	83.09	76	92.58	77.44	70.04	89.2
QNN HTP V79 (5K)	84.41	78.2	93.66	82.6	75.7	92.0	76.95	69.4	88.42
QNN HTP V75 (5K)	84.41	78.2	93.66	82.6	75.7	92.0	76.95	69.4	88.42
QNN HTP V73 (5K)	84.54	78.24	93.58	82.69	75.54	92.1	76.95	69.56	88.4
QNN HTP V69 (5K)	84.66	78.46	93.68	82.63	75.64	92.14	76.93	69.6	88.36
QNN HTP V68 (5K)	84.66	78.46	93.68	82.63	75.64	92.14	76.93	69.6	88.36
QNN GPU-16 (5K)	84.96	78.62	94.18	83.09	76.02	92.6	77.46	70.06	89.24
QNN GPU-32 (5K)	84.97	78.56	94.18	83.07	76	92.6	77.37	70.04	89.28
QNN DSP-V66 (5K)	84.52	78.34	93.66	82.58	75.52	92	76.87	69.42	88.44
QNN HTA (5K)	84.39	78.62	93.22	82.5	75.62	91.7	76.78	69.36	88.2

NOTES:

- Top-1 and Top-5 represent the accuracy percentage on the most likely and 5-most-likely predicted classes, on average
- For the mAP definition, and for an explanation of the measurement methodology see the “Precision Measurement” application note
- Image classification validation on **5k images** of ImageNet 2012 (ILSVRC2015/CLS_LOC/Data/CLS-LOC/val)
- Values are measured on a variety of devices and are consistent across devices.
- HTP, DSP, HTA numbers are calculated using quantized Models with 8bit input format.

Performance Measurement Environment

Performance measurement environment

- Runtime support:
  - CPU ☑ , GPU ☑ DSP ☑, AIP ☑

In an effort to provide performance metrics that are more similar to the values that would be seen in a SNPE deployment, the environment used to run the benchmarks has been updated.

- The MTP and QRD Devices use the Secondary Boot image, as this is more representative of the OS image used in a commercial device
- The benchmarking is done using the ‘basic’ diagnostics profile, as this is more representative of a commercial deployment (which likely will not want to capture intermediate performance metrics).
- These two changes improve the reported values due to the reduced overhead of the runtime environment:
  - Switching to the Secondary Boot Image improves the inf/s in the general range of 4-12%
  - Using the ‘basic’ Diagnostics Profile improves the inf/s in the general range of 1-2%

Note: The change in performance is highly dependent on the network, runtime, and other system settings.

Performance Measurement Methodology

Performance measurement methodology

- Runtime support:
  - CPU ☑ , GPU ☑
    DSP ☑, AIP ☑

The following method is used to measure the performance of different models.

- Number of images used : 100 per model
- Performance Profile :
  - High_performance : CPU, GPU
  - Burst : DSP, AIP
- Profiling Level : basic
  - Basic profiling is enabled for CPU, DSP and AIP runtimes and GPU measurements are with detailed profiling.

- Execution :
  - Three runs of snpe-net-run with 100 images per run.

- Calculation :
  - Total inference time is averaged over the 3 Runs.
  - **Inf/sec** = (10^6 / Total Inference Time) * Batch_Size

- Example:
  - `python snpe_bench.py -c config.json -p high_performance -l basic -o output_dir`

Image Classification Precision Measurement

Details on precision measurement for image classification networks.

- Our measurements are repeated across chipsets; accuracy scores do not vary per chipset
- Target acceleration cores: GPU (FP32), DSP (INT8), CPU (baseline comparison)
- Report generated for:
  - Networks: Inception_v3_2016, Resnet50, VGG16, Yolo_v5, Mobilenet_v3_minimal
  - Validation set: ImageNet 2012, in the *ILSVRC2012/CLS_LOC/Data/CLS-LOC/val* folder
- Qualcomm precision metrics:
  - mAP: mean average precision across all categories
  - Top-1 error rate: chance the highest-probability predicted class is not the real class
  - Top-5 error rate: chance the real class is not contained in the 5 classes with highest probability

Image Classification Precision Measurement

Precision Measurement Methodology

1. Various devices (one per major DNN) loaded with all 50000 images from the ImageNet 2012 validation set preprocessed as prescribed by the paper or official documentation of each model

2. Run these images through the model using SNPE, using snpe-net-run and testing all available runtimes: CPU, GPU, DSP

3. Compare the output of SNPE with ground truth (ImageNet labels) and compute mAP, Top-1 Error and Top-5 Error

Image Classification Precision Measurement

Precision Measurement Metrics

AP, mAP:

$$precision = \frac{tp}{tp + fp} \qquad recall = \frac{tp}{tp + fn}$$

- Where: tp = true positives; fp = false positives; fn = false negatives
- Average Precision (AP) is calculated per category

- mAP (mean Average Precision) is the average precision across all categories

Top-1 Error, Top-5 Error:

- Rank-N error is the fraction of test samples x_i where the correct label y_i does not appear in the top N predicted results of the model when results are sorted in decreasing order of confidence
- For an example usage, see this paper: https://arxiv.org/pdf/1409.0575.pdf

Documentation Addendum

Additional documentation that will be updated in SDK html documentation in upcoming releases

Index

Change in quantizer behavior for Tensorflow QuantAware Training networks

added in 2.13.0

The quantization aware trained (QAT) tensorflow networks carry the quantization data through fakequant nodes. As per documentation, the quantization data embedded in source model should be used against the quantization data generated by internal QNN quantizer using calibration dataset (i.e., input_list). A bug is fixed to apply the above behavior in current release.

The following behaviors are expected for tensorflow networks with fakequant nodes:

1. Change in quantization params for tensors having fakequant nodes in DLC. Some networks may exhibit change in accuracy metric scores.
   - Mitigation plan: Disable the feature by dropping “override_params” to snpe-dlc-quant
2. Networks contain INT16 bitwidth QAT encodings start fail in inference for CPU and DSP runtimes as they don’t natively support INT16
   - Possible error messages:
     [USER_WARNING] Validation for Op=<xxxx> Type=<xxxx> at pos=<xx> failed for runtime=<xx>
   - Mitigation plan: Disable the feature by dropping “override_params” to snpe-dlc-quant
3. Networks contain QAT encodings with bitwidths other than supported(8, 16) start fail in conversion
   - Error message:
     [ERROR] getDataTypeFixedPoint: Invalid bitwidth: 10
   - Mitigation plan: Disable the feature by dropping “override_params” to snpe-dlc-quant

Workaround for snpe quantization failures

added in 2.13.0

SNPE quantizer (snpe-dlc-quant) may fail with following error message for networks having input tensors with special characters.

[ERROR] **Error processing inputs to graph. Check input data paths.**

Example:

- Order of input tensor names as per DLC info: abc:0, xyz:0
- Order of input tensor names specified in input_list.txt: xyz:0, abc:0
  - Quantizer may fail when the above input_list.txt is specified to snpe-dlc-quant
- Mitigation plan: Provide the raw filenames in the order of inputs described in DLC info.

Converter and Quantizer commandline Argument Deprecation

added in 2.18.0

- Some arguments of SNPE Converters and Quantizer are deprecated in this release.
- A warning message will be printed when such deprecated argument is used.
- The deprecated arguments will be removed in a future release.
- **Deprecated SNPE Converter Commandline Arguments:**
  - The following arguments are obsolete.

Argument	Mitigation/Alternative Argument
--validation_target	Obsolete. Deprecated.
--strict	Obsolete. Deprecated.

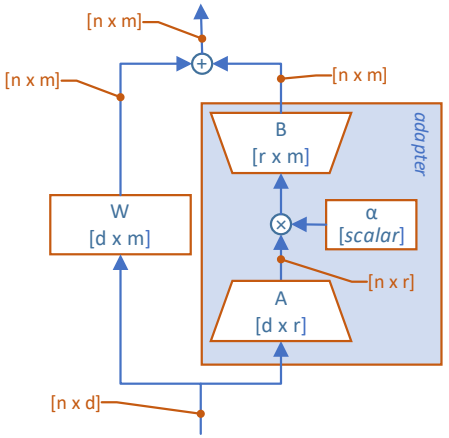
- **Deprecated SNPE Quantizer Commandline Arguments:**
  - New alternative arguments are added for some of the deprecated arguments to enable the same functionality.

Argument	Mitigation/Alternative Argument
--no_weight_quantization	Obsolete. Deprecated.
--bitwidth	Use –act_bitwidth, --weights_bitwidth, --bias_width options for specifying values for different tensor types.
--clip_alpha	Obsolete. Deprecated.
--axis_quant	Use “--use_per_channel_quantization”
--use_enhanced_quantizer --use_adjusted_quantizer --use_symmetric_quantize_weights	Use “–param_quantizer” and “–act_quantizer” to specify appropriate quantization schemes.
--optimizations	Use “–algorithms” to enable the same functionality
--float_bw	Use “–float_bitwidth”
--use_encoding_optimizations	Obsolete. Deprecated.

SM8750 HTP and GPU Clock Frequency

Performance Profile	GPU OpenCL Macro	GPU Frequency (MHz)	HTP DCVS Macro	HVX Frequency (MHz)	HMX Frequency (MHz)
BURST	CL_PERF_HINT_HIGH_QCOM	1100	DCVS_VOLTAGE_VCORNER_MAX_VOLTAGE_CORNER	2112	1209.6
HIGH_PERFORMANCE	CL_PERF_HINT_HIGH_QCOM	1100	DCVS_VOLTAGE_VCORNER_TURBO	1593.6	979.2
SUSTAINED_HIGH_PERFORMANCE	CL_PERF_HINT_HIGH_QCOM	1100	DCVS_VOLTAGE_VCORNER_TURBO	1593.6	979.2
DEFAULT	CL_PERF_HINT_HIGH_QCOM	1100	DCVS_VOLTAGE_VCORNER_NOM_PLUS	1382.4	806.2
BALANCED	CL_PERF_HINT_NORMAL_QCOM	160-1100	DCVS_VOLTAGE_VCORNER_NOM_PLUS	1382.4	806.2
LOW_BALANCED	CL_PERF_HINT_LOW_QCOM	160	DCVS_VOLTAGE_VCORNER_NOM	1248	672
HIGH_POWER_SAVER	CL_PERF_HINT_LOW_QCOM	160	DCVS_VOLTAGE_VCORNER_SVS_PLUS	979.2	499.2
POWER_SAVER	CL_PERF_HINT_LOW_QCOM	160	DCVS_VOLTAGE_VCORNER_SVS	806.4	499.2
LOW_POWER_SAVER	CL_PERF_HINT_LOW_QCOM	160	DCVS_VOLTAGE_VCORNER_SVS2	576	364.8
EXTREME_POWER_SAVER	CL_PERF_HINT_LOW_QCOM	160	DCVS_VOLTAGE_CORNER_DISABLE	384	249.6

LoRA Introduction



- LoRA = Low Rank Adaptation

- LoRA v1 : LoRA weights as inputs
  - LoRA supported by transforming the model and applying LoRA weights as inputs

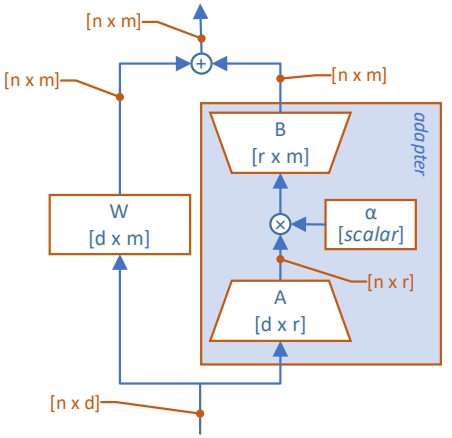
- LoRA v2 : Native LoRA
  - LoRA weights are applied “natively” into the graph as weights at attachment points
  - Advantages : performance, regained accuracy on Quantization

- LoRA Terminology :
  - Adapter : set of LoRA weights for one use case
  - Alpha : parameter for setting the strength

LoRA v2 Overview

Adapter Requirements :

- PEFT-based Adapters (PEFT = Parameter-Efficient Fine-Tuning)
- All Adapters to a given base graph :
  1. Have same architecture and attachment points
  2. Have the same max rank and same precision

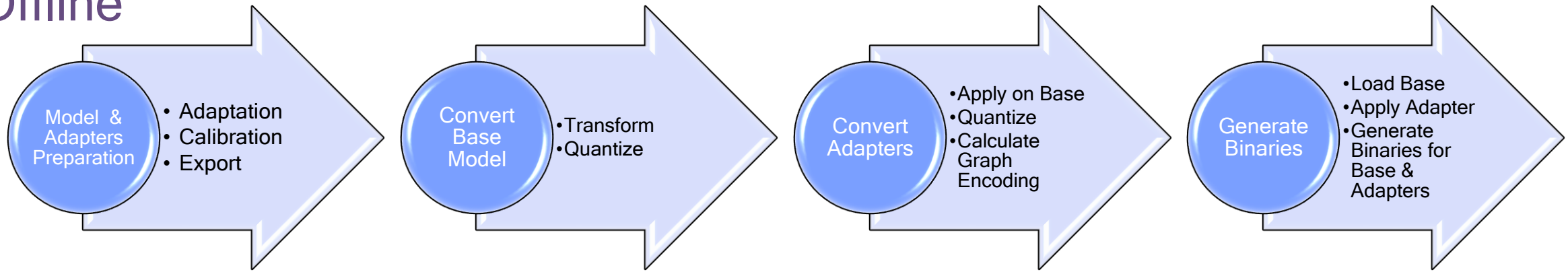


Supported Functionality :

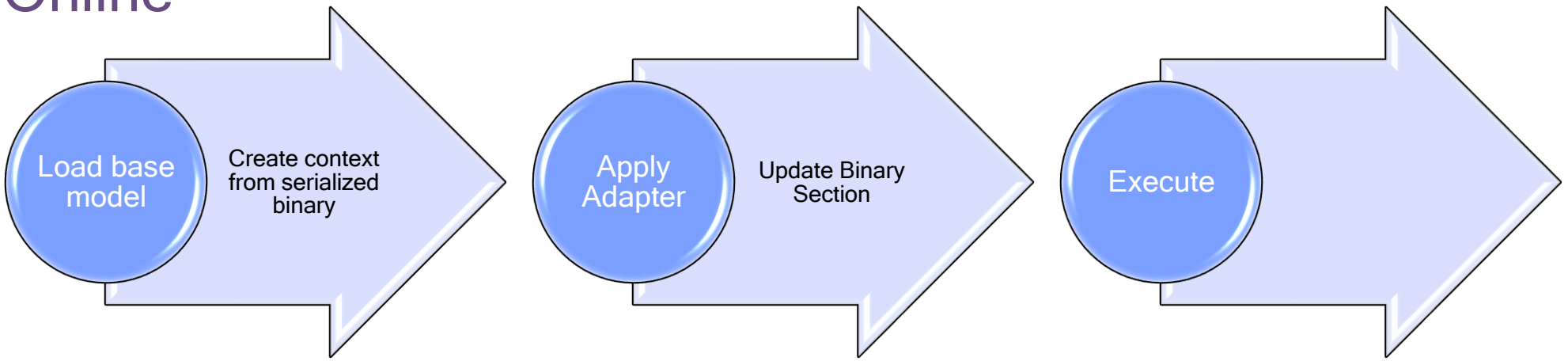
- Currently supported for ONNX models and HTP backend only
- Apply a single adapter at a given time (e.g single branch)
- Dynamic switching of adapter (e.g. without loading/unloading of base model)
- Regain accuracy on Quantization by creating tailored encodings per adapter during offline conversion
  - Adapter weights are quantized, and each weight has its own encodings
  - Activation encodings for each adapter are different and optimized during calibration process
- Requires full offline preparation of both Base Model & Adapters, with Quantization done by AIMET
  - Offline preparation only supported on :Linux x86 host platforms
  - **Disclaimer: Both base model context binary and adapter binary files MUST be prepared using same QAIRT SDK version.**

High-Level end-to-end workflow

Offline



Online

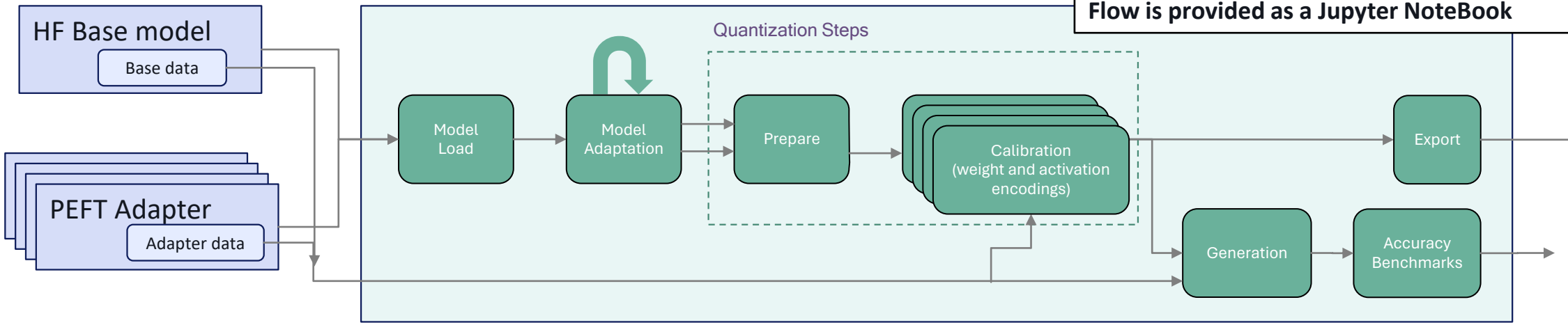


Notes :

1. For switching adapters, “Apply adapter” can be done as needed between calls to execute
2. Setting Alpha is done by updating the relevant input tensor

Offline Flow : Model Preparation

AIMET: Model Prep and Quant, and Export to ONNX.
Flow is provided as a Jupyter NoteBook

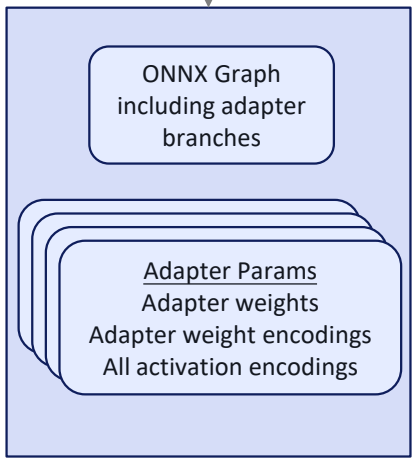


Inputs:

- Base models as a python script/notebook file and .safetensors checkpoint
- Bitwidth specification for the base model
- Set of calibration data (correct format for model input)
- For each adapter
  - Adapter specified as .safetensors checkpoint supporting PEFT load onto base model
  - PEFT config is available in the combined model, including adapter ranks
  - Bitwidths of adapter branches
  - Set of calibration data (correct format for model input) relevant for the adapter
  - All adapters are required (and checked) to have the same rank, attachments, and bitwidths

Outputs for QAIRT use:

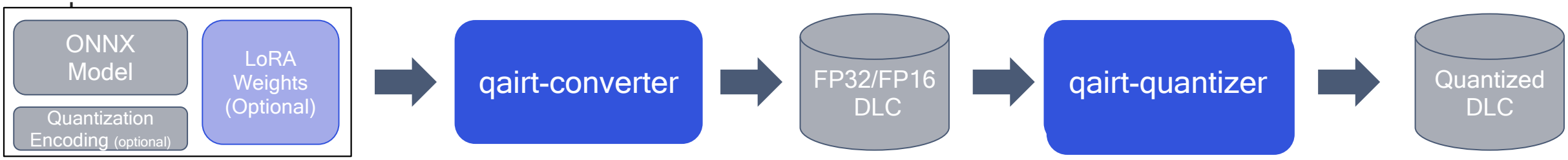
- ONNX graph
  - including adapter branches (same for all adapters)
  - adaptations applied
- For each adapter
  - adapter weights file (.safetensors)
  - quantization encodings file (.encoding), holding encodings for both adapter weights & all activations



Offline Flow : Conversion

Convert the Base Model

- Same Converter/Quantizer for LoRA and non-LoRA usages
- Converter has a new “--lora_weight_list” parameter to pass LoRA Weights Names, to identify as “updatable tensors” in the



Import the LoRA Adapters (new tool : qairt-lora-importer)

Note: This feature is currently supported only for ONNX models

Offline Flow : Conversion

What is the format of “LoRA Weights Names” provided to qairt-converter tool with --lora_weight_list option ?

- A text file contains LoRA adapter weight tensor names with newline as delimiter
- Refer to the below sample code that generates a text file with LoRA adapter weight tensor names from LoRA adapter .safetensors file

```
from safetensors.numpy import load_file

def save_tensor_names(safetensor_path, save_path="./tensor_names.txt"):
    tensor_name_to_data_map = load_file(safetensor_path)
    with open(save_path, 'w') as text_file:
        for tensorAtt_name in tensor_name_to_data_map:
            text_file.write(tensor_name + '\n')
```

Offline Flow : Conversion

What is the format of “LoRA Adapter Config File” provided to qairt-lora-importer tool with --lora_config option?

- A YAML file contains artifacts of LoRA use-cases (downstream tasks)
- Sample example:

```
##################################
# LoRA YAML config
##################################

# Start config
  use_case:   # List of use-cases (Downstream tasks)
      # lora_weights and  quant_overrides files must have weight tensor names in them as seen in onnx file (model_name).
    - name:                 <usecase1name>
      model_name:           <base model.onnx>
      lora_weights:         <usecase1name.safetensors>
      quant_overrides:      <usecase1name.encodings>
      output_path:          <output folder>
    - name:                 <usecase2name>
      model_name:           <base model.onnx>
      lora_weights:         <usecase2name.safetensors>
      quant_overrides:      <usecase2name.encodings>
      output_path:          <output folder>

# End config
```

Offline Flow : Conversion

Sample Converter, Quantizer and LoRA Importer commands

- qairt-converter command:

```
qairt-converter -i <model.onnx> --quantization_overrides <base_model.encodings> --lora_weight_list <tensor_names_from_safetensor_file.txt>
```

- qairt-quantizer command:

Using Input List:

```
qairt-quantizer --input_dlc <path to dlc produced with lora> --input_list <input_list.txt>

Note: Please ensure input_list.txt follows the below format when graph has multiple input tensors.
<input_name>:=<input_raw_file_path> <input_name>:=<input_raw_file_path> <input_name>:=<input_raw_file_path>
```

Using FloatFallback:

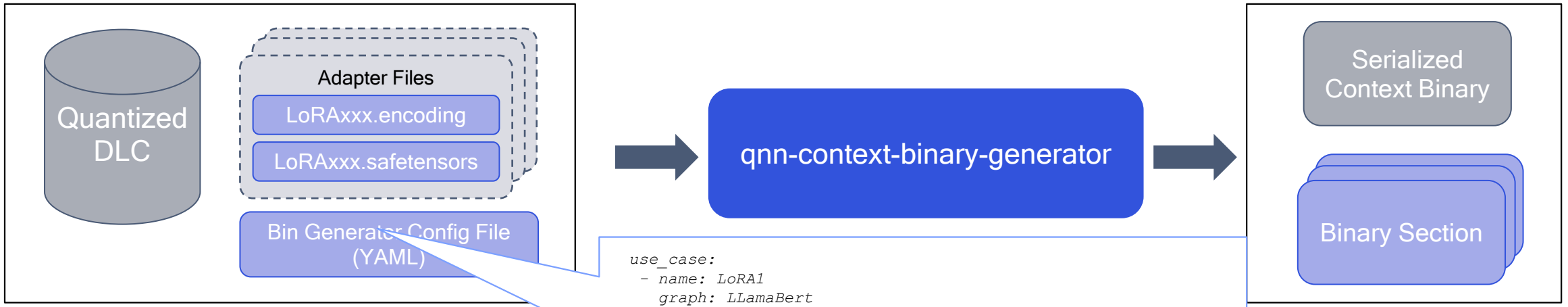
```
qairt-quantizer --input_dlc <path to dlc produced with lora> --float_fallback
```

- qairt-lora-importer command:

```
qairt-lora-importer -i <model.onnx> --lora_config <lora_config.yaml> --input_dlc <path to quantized/unquantized lora dlc>  --input_list <input_list.txt>
```

Offline Flow : Generating Binaries

- qnn-context-binary-generator is extended to support Applying of LoRA weights :
  - Receive a new *"--adapter_weight_config"* parameter to receive Adapters YAML config file (produced by qairt-lora-importer tool)
  - Generate a "binary section" file for each LoRA adapter
- The produced "binary section" file is used on-target with new QNN API to apply the LoRA adapter
  - Notice: The QAIRT SDK version that generates the base graph context binary and the adapter binary file MUST be the same.



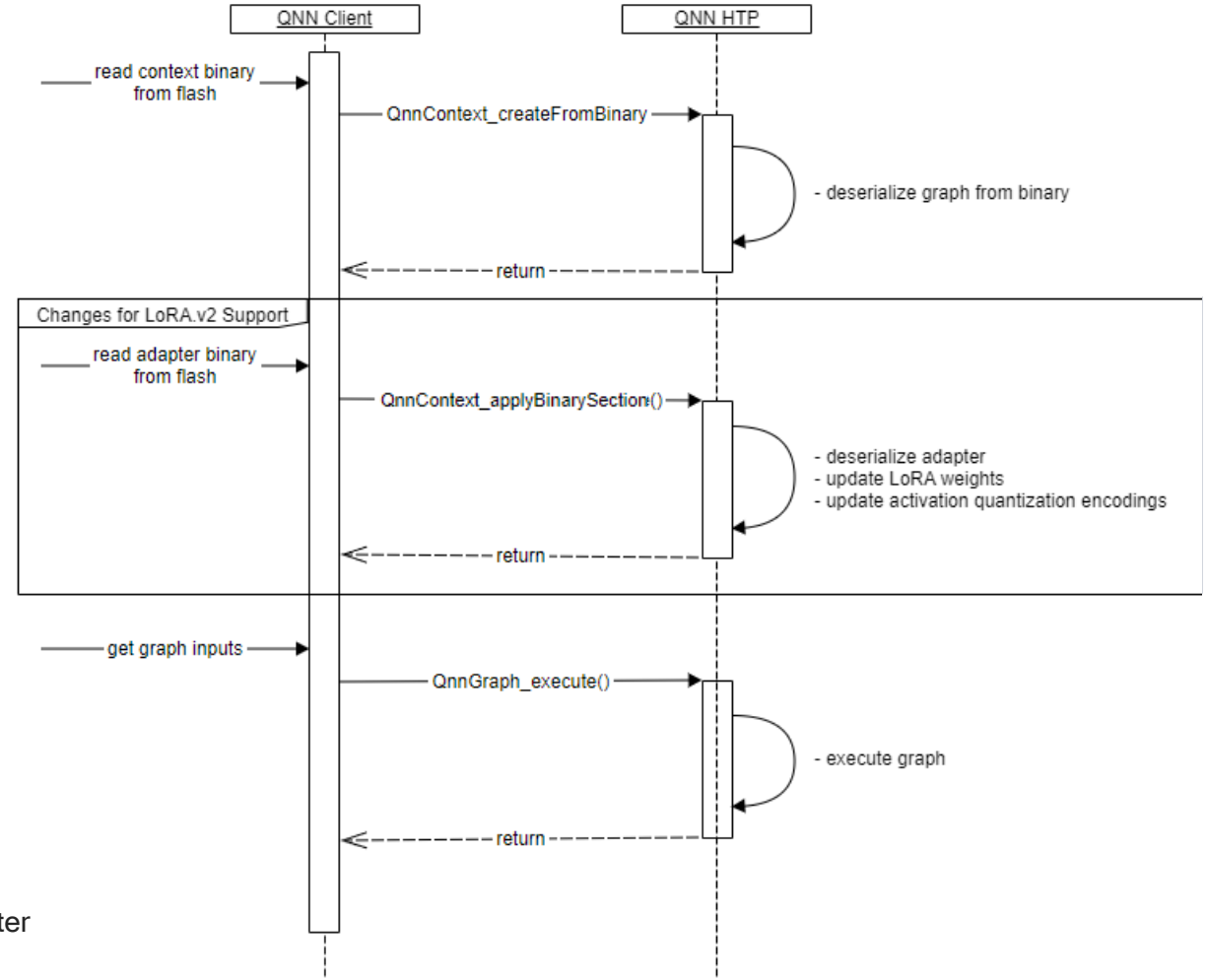

```
use_case:
 - name: LoRA1
   graph: LLamaBert
   weights:LoRA1.safetensors
   encodings: LoRA1.encodings
 - name: LoRA2
   graph: LLamaBert
   weights: LoRA2.safetensors
   encodings: LoRA2.encodings
share_adapters_between_graphs: Yes # added in 2.25.15
...
```

share_adapters_between_graphs would produce fewer adapter files - Instead of 1 adapter per graph, it would be 1 per group of graphs that are sharing weights. The combined adapter file, also referred to as "Super Adapter", enables weight sharing between adapters, thereby reducing total adapter size.

Online Flow : QNN Call Flow

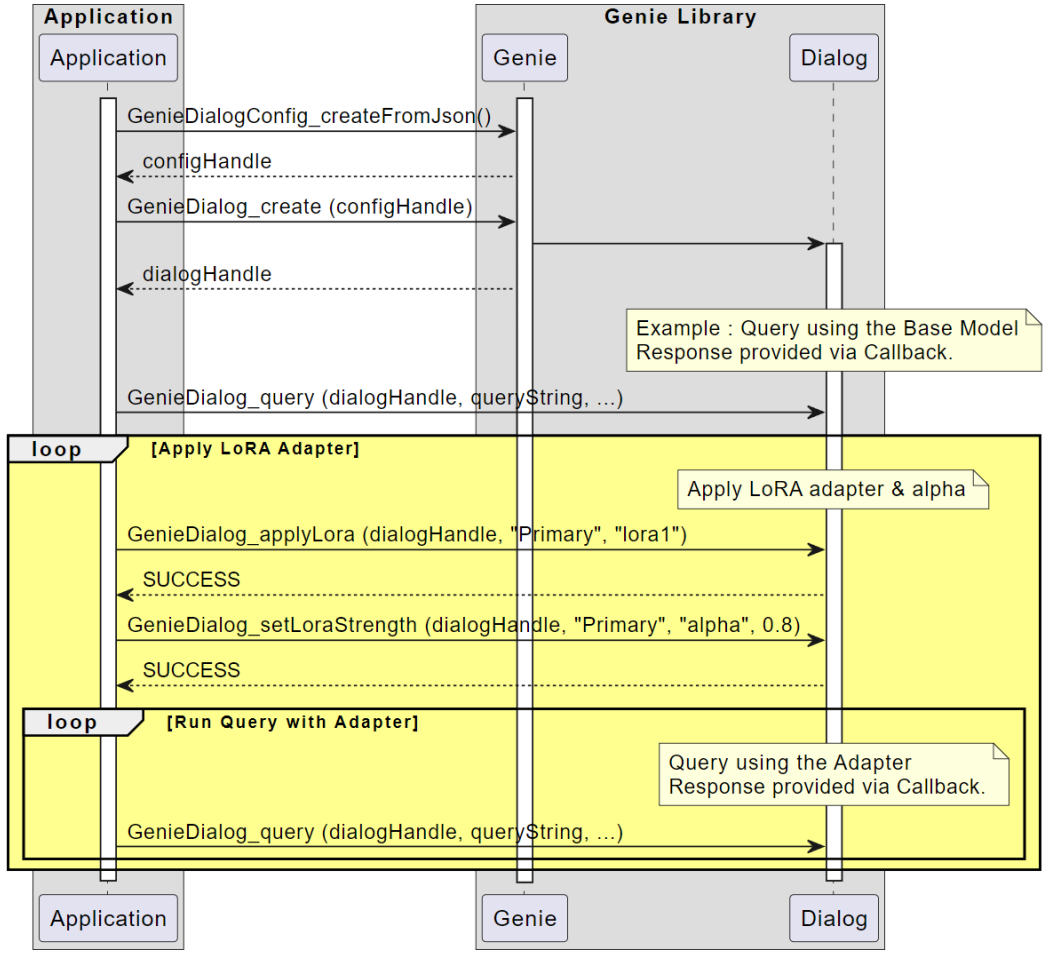
- At the end of the offline flow, users will have a serialized context binary file (for base model) , and a set of binary section files (for LoRA Adapters)
- In order to apply LoRA Adapter on-target, user needs to use new QNN API: QnnContext_applyBinarySection( )
- The on-target flow is as follows :
  - Create Context by calling QnnContext_createFromBinary (as usual)
  - Apply the adapter by calling QnnContext_applyBinarySection (new)
  - Get inputs and call QnnGraph_execute (as always)
  - When need to apply a different Adapter :
  - Call QnnContext_applyBinarySection again with a different binary section file
  - Get inputs and call QnnGraph_execute.
- Back to running with base graph only (after any adapter is applied)
  - Option a - Set Alpha to 0
  - Option b - Create one adapter which has all zero Lora weights. Switch to this adapter.

    A default adapter is generated from the qnn-context-binary-generator with suffix default_adapter



Online Flow : Genie LoRA API

- Genie library provides high-level Dialog API for Generative AI transformer models
- Dialog API is extended to include applying a LoRA adapter and setting the strength (alpha)

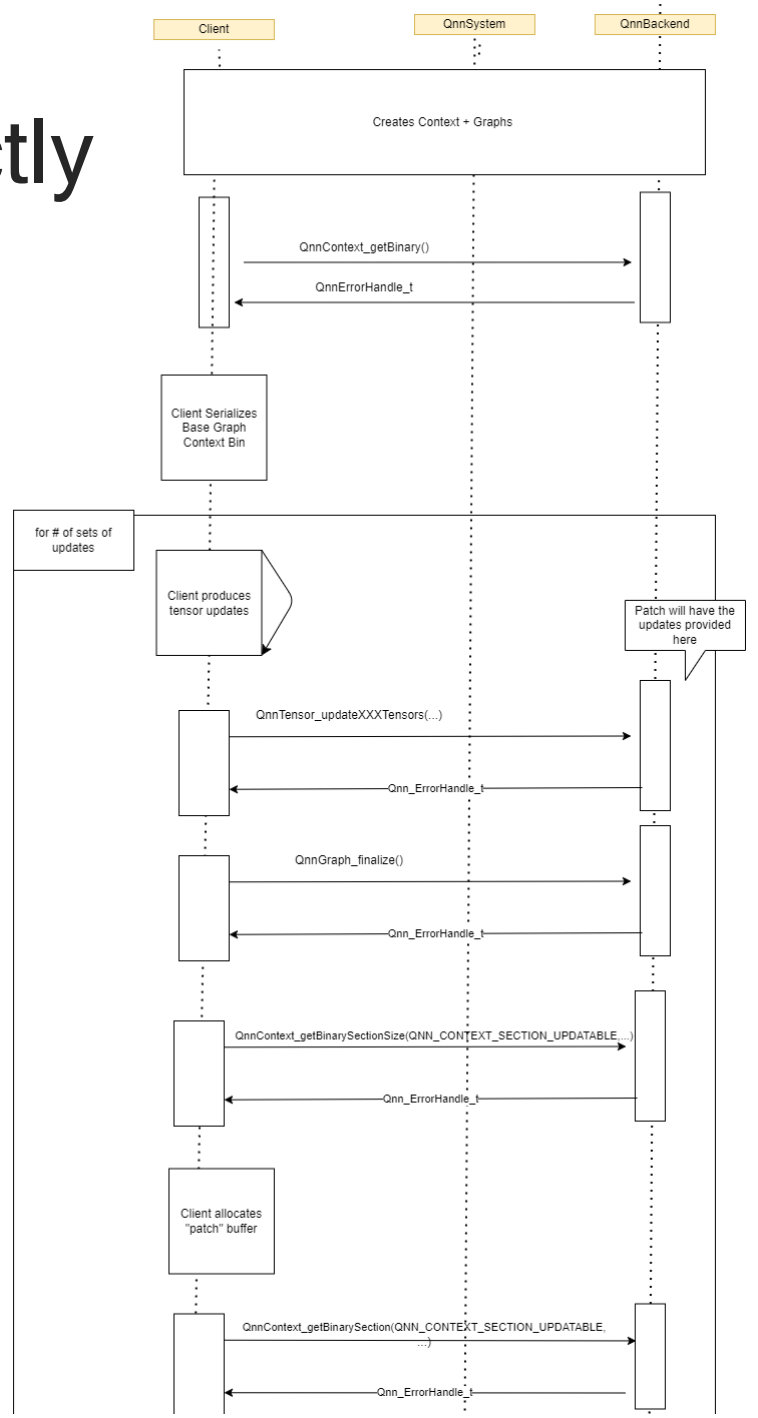


Example : LoRA JSON configuration
(section of the larger dialog JSON config)

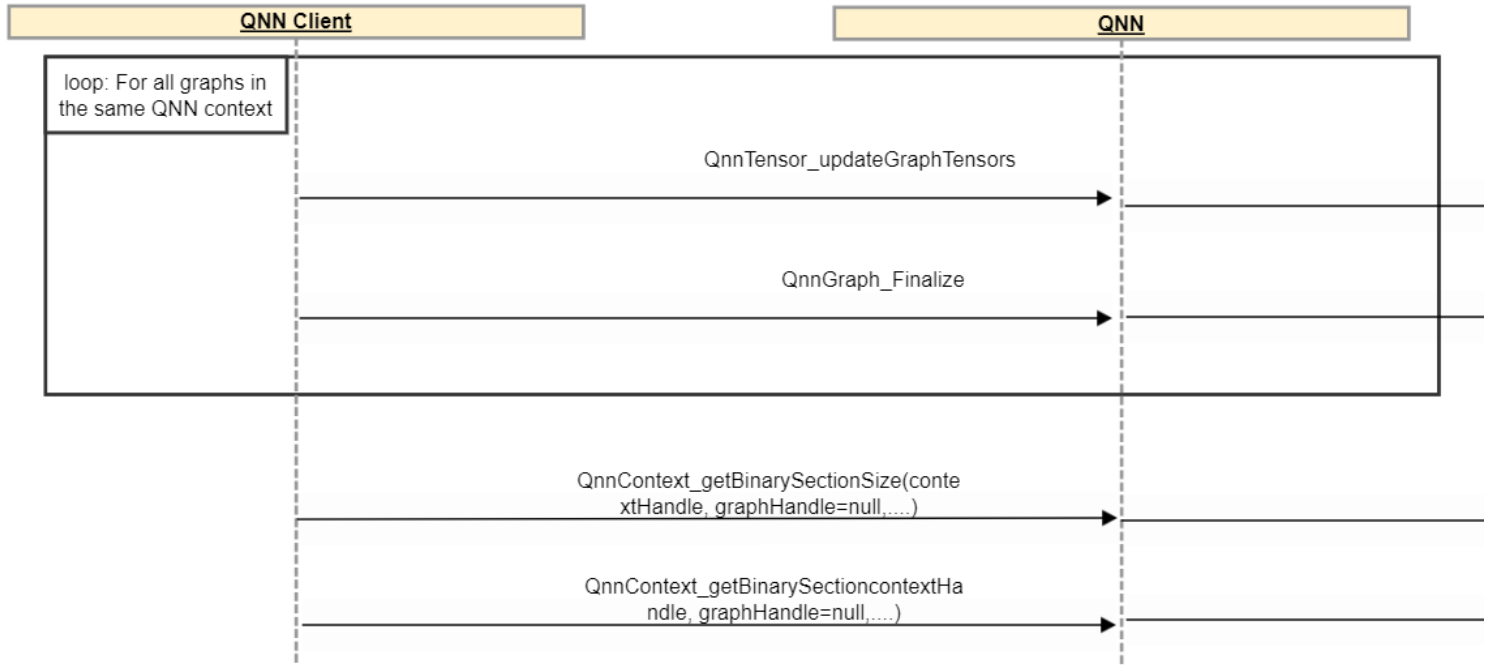
```
{
 "lora": [
    {
      "adapter-name" : "lora1",
      "alpha-tensor-name": "alpha",
      "alpha-tensor-value": 0.3,
      "binsection-basedir": "lorav2/binsections",
      "bin-sections": [
        "Lorabinsection1.bin",
        "Lorabinsection2.bin",
        "Lorabinsection3.bin",
        "Lorabinsection4.bin"
      ]
    },
    {
      "adapter-name" : "lora2",
      "alpha-tensor-name": "alpha",
      "alpha-tensor-value": 0.4,
      "binsection-basedir": "lorav22/binsections",
      "bin-sections": [
        "Lorabinsection21.bin",
        "Lorabinsection22.bin",
        "Lorabinsection23.bin",
        "Lorabinsection24.bin"
      ]
    },
    ...
 ]
}
```

Generating the LoRA binary sections directly

- As shown on previous slides, *"qnn-context-binary-generator"* tool is extended to produce LoRA binary sections
- This slide explains how do apply the adapter weights and retrieve the binary sections directly using QNN API (not via QNN tools)
- This is done in the following manner :
  1. Create the QNN Context & Graphs (either from-scratch or from a Binary)
     a. In case the Context/Graph was created from scratch, Call *QnnContext_getBinary* to receive a binary blob of the unmodified QNN context.
  2. Call New QNN API : *QnnTensor_updateGraphTensors / QnnTensor_updateContextTensors*
     a. Tensors must be of UPDATEABLE type, created during graph composition (in step 1.)
  3. Call *QnnGraph_finalize* (**important !** Updates are not applied until finalize is called)
  4. Call *QnnContext_getBinarySectionSize* to receive the size of the binary section
     a. A buffer with a suitable size should be allocated and passed to QNN backend as part of the next API call
  5. Call *QnnContext_getBinarySection* to receive a binary blob containing the LoRA update
- Steps 2-4 can be done multiple times, each time apply a different adapter (by updating the weights) and retrieve a suitable binary section



Generating LoRA weight-shared binary sections directly



- There is a slight call-flow change when generating weight-shared binary sections (Super Adapters).
- Compared to the the previous slide, where every *QNNGraph_Finalize* is followed by *QnnContext_getBinarySectionSize* then *QnnContext_getBinarySection,* get Binary Sections is called once per a group of graphs that are in the same context. The combined adapter file, also referred to as “Super Adapter”, enables weight sharing between adapters, thereby reducing total adapter size.
- As mentioned in previous slides, Super Adapters are generated through *qnn-context-binary-generator* by adding following option in Config YAML File:
  - ***share_adapters_between_graphs: Yes***

LoRA + Graph switch Implementation

- Graph switching can now be used with LoRA to reduce RAM usage by trading off slight token rate hit
- As per QNN SDK doc, to enable graph switch, user needs to set context config options as
  `QNN_CONTEXT_CONFIG_MEMORY_LIMIT_HINT` : **non-zero value**
  `QNN_CONTEXT_CONFIG_PERSISTENT_BINARY` : **true**
- If user uses qnn-net-run or qnn-throughput-net-run, this can be done by setting config options:
  `memory_limit_hint`
  `is_persistent_binary`
  accordingly in backend extension config file.
- The adapter buffer should be kept persistent (like context binary buffer) for graph switching
  - During *QnnContext_applyBinarySection*, if the graph is in an unloaded state, HTP Backend deserializes the graph, then applies the adapter.
  - During *QnnGraph_execute*, if the graph is in an unloaded state, HTP backend loads the unloaded graph and then reapplies last applied adapter from the persistent buffer.

New QNN API signatures related to LoRA

Next few slides provide the signatures of the new QNN API calls :

- QnnTensor_updateGraphTensors
- QnnTensor_updateContextTensors
- QnnContext_getBinarySectionSize
- QnnContext_getBinarySection
- QnnContext_applyBinarySection

New QNN API :

QnnTensor_updateGraphTensors/ QnnTensor_updateContextTensors

```
/**
 * @brief Update a graph tensor with the new provided tensor information.
 *        Tensors provided here are associated with the tensor in the backend through the ID field.
 *        Valid fields to update are: data and quantization parameters for UPDATEABLE_STATIC tensors,
 *        quantization parameters for UPDATEABLE_NATIVE, UPDATEABLE_APP_READ, UPDATEABLE_APP_WRITE,
 *        and UPDATEABLE_APP_READWRITE tensors.
 *        Multiple calls to QnnTensor_updateGraphTensors() can be made, but the updates will
 *        not take effect until QnnGraph_finalize() is called.
 *        Backends may support a subset of updateable tensor types.
 *
 ... (error codes)
*/
QNN_API
Qnn_ErrorHandle_t QnnTensor_updateGraphTensors(Qnn_GraphHandle_t graph,
                                               const Qnn_Tensor_t** tensor,
                                               uint64_t numTensors);

/**
 * @brief Update a context tensor with the new provided tensor information.
 *        Tensors provided here are associated with the tensor in the backend through the ID field.
 *        Valid fields to update are: data and quantization parameters for UPDATEABLE_STATIC tensors,
 *        quantization parameters for UPDATEABLE_NATIVE, UPDATEABLE_APP_READ, UPDATEABLE_APP_WRITE,
 *        and UPDATEABLE_APP_READWRITE tensors.
 *        Multiple calls to QnnTensor_updateContextTensors() can be made, but the updates will
 *        not take effect until QnnGraph_finalize() is called for one or more of the graphs to
 *        which the context tensors are associated.
 *        Backends may support a subset of updateable tensor types.
 *
 ... (error codes)
*/
QNN_API
Qnn_ErrorHandle_t QnnTensor_updateContextTensors(Qnn_ContextHandle_t context,
                                                 const Qnn_Tensor_t** tensor,
                                                 uint64_t numTensors);
```

New QNN API : QnnContext_getBinarySectionSize

```
/**
 * @brief Retrieve a section of the binary as specified by __section__. The size of this section
 *        depends on the type of section requested. For example, for QNN_CONTEXT_SECTION_UPDATABLE
 *        sections, this will have all the updatable tensor information.
 *
 * @param[in] context A context handle.
 *
 * @param[in] graph A graph handle. This argument is optional. When supplied the return size only
 *                  applies to the size of the context binary section pertaining to this graph. When
 *                  excluded the returned binary contains associated updates to all graphs in the
 *                  context. Some backends may require _graph_ as an argument. Support is determined
 *                  by QNN_PROPERTY_CONTEXT_SUPPORT_BINARY_SECTION_FULL_CONTEXT.
 *
 * @param[in] section The section of the binary to retrieve.
 *
 * @param[out] binaryBufferSize The amount of memory in bytes a client will need to allocate
 *                              to hold context content updates in binary form.
 *
 ... (error codes)
*/

QNN_API
Qnn_ErrorHandle_t QnnContext_getBinarySectionSize(Qnn_ContextHandle_t context,
                                                  Qnn_GraphHandle_t graph,
                                                  QnnContext_SectionType_t section,
                                                  Qnn_ContextBinarySize_t* binaryBufferSize);
```

New QNN API : QnnContext_getBinarySection

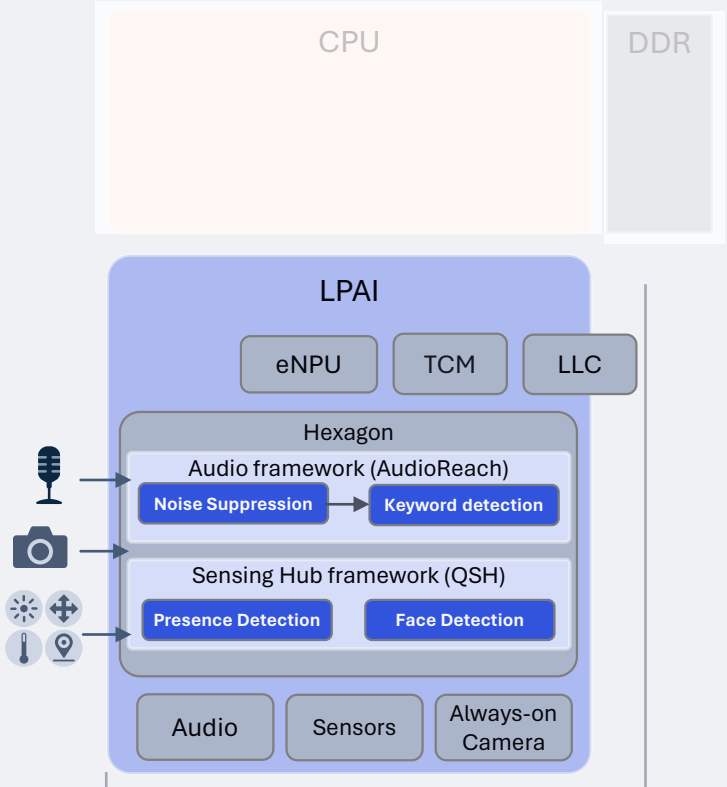
```
/**
 * @brief Retrieve section of the context binary. Content of the section is specified by
 *        __section__. The size of the section is retrieved from QnnContext_getBinarySectionSize().
 *
 * @param[in] context A context handle.
 *
 * @param[in] binaryBuffer Pointer to the user-allocated context binary memory.
 *
 * @param[in] graph A graph handle. This argument is optional. When supplied the returned binary
 *                  only contains the context binary section pertaining to this graph. When excluded
 *                  the returned binary contains associated updates to all graphs in the context.
 *                  Some backends may require _graph_ as an argument. Support is determined by
 *                  QNN_PROPERTY_CONTEXT_SUPPORT_BINARY_SECTION_FULL_CONTEXT.
 *
 * @param[in] section The section of the binary to retrieve. When section is
 *                    QNN_CONTEXT_SECTION_UPDATABLE the returned binary will contain all of the
 *                    updatable tensors associated with the context and graph combination.
 *
 * @param[in] profile The profile handle on which metrics are populated and can be queried. Use
 *                    NULL handle to disable profile collection. A handle being re-used would reset
 *                    and is populated with values from the current call.
 *
 * @param[in] signal Signal object to control the execution of the create context from binary
 *                   process. NULL may be passed to indicate that no execution control is requested,
 *                   and the create operation should continue to completion uninterrupted.
 *                   The signal object, if not NULL, is considered to be in-use for
 *                   the duration of the call.
 *
 * @param[out] writtenBufferSize Amount of memory actually written into _binaryBuffer_, in bytes.
 *
*/
QNN_API
Qnn_ErrorHandle_t QnnContext_getBinarySection(Qnn_ContextHandle_t context,
                                              Qnn_GraphHandle_t graph,
                                              QnnContext_SectionType_t section,
                                              const QnnContext_Buffer_t* binaryBuffer,
                                              Qnn_ContextBinarySize_t* writtenBufferSize,
                                              Qnn_ProfileHandle_t profile,
                                              Qnn_SignalHandle_t signal);
```

New QNN API : QnnContext_applyBinarySection

```
/**
 * @brief Apply a section to the contextBinary produced by a prior QnnContext_getBinarySection()
 *        call. If successful, this section overwrites previously applied sections. If the call to
 *        applyBinarySection() fails, it indicates the changes were not applied, and that the
 *        context retains its prior state. In this case the context is still valid and may be used
 *        for subsequent inferences.
 *
 * @param[in] context A context handle.
 *
 * @param[in] graph A graph handle. This argument is optional. When supplied the returned binary
 *                  only contains the context binary section pertaining to this graph. When excluded
 *                  the returned binary contains associated updates to all graphs in the context.
 *
 * @param[in] section The section of the binary to retrieve. When section is
 *                    QNN_CONTEXT_SECTION_UPDATABLE the returned binary will contain all of the
 *                    updatable tensors associated with the context and graph combination.
 *
 * @param[in] binaryBuffer Pointer to the user-allocated context binary memory.
 *
 * @param[in] profile The profile handle on which metrics are populated and can be queried. Use
 *                    NULL handle to disable profile collection. A handle being re-used would reset
 *                    and is populated with values from the current call.
 *
 * @param[in] signal Signal object to control the execution of the create context from binary
 *                   process. NULL may be passed to indicate that no execution control is requested,
 *                   and the create operation should continue to completion uninterrupted.
 *                   The signal object, if not NULL, is considered to be in-use for
 *                   the duration of the call.
 *
 ... (error codes)
*/
QNN_API
Qnn_ErrorHandle_t QnnContext_applyBinarySection(Qnn_ContextHandle_t context,
                                                Qnn_GraphHandle_t graph,
                                                QnnContext_SectionType_t section,
                                                const QnnContext_Buffer_t* binaryBuffer,
                                                Qnn_ProfileHandle_t profile,
                                                Qnn_SignalHandle_t signal);
```

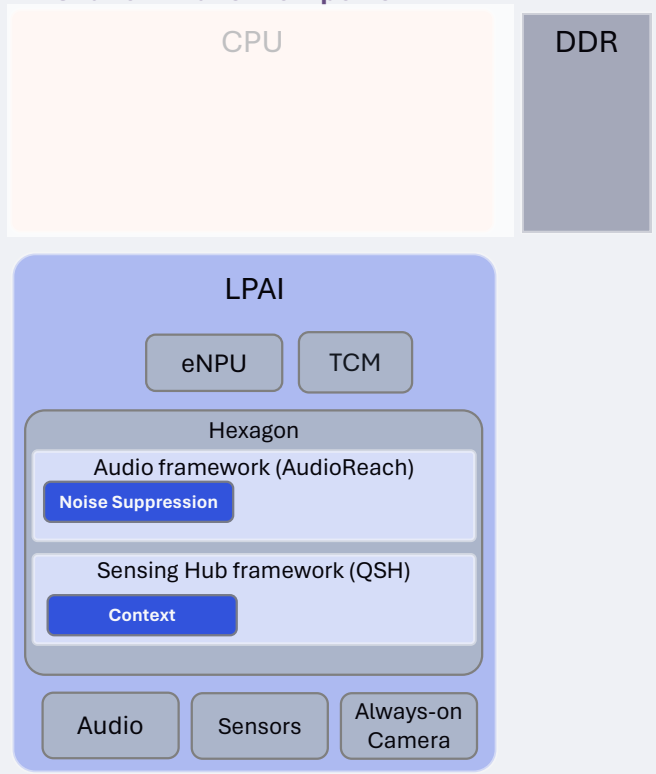
LPAI : Different Modes of Operation

Ultra Low Power (LPI)

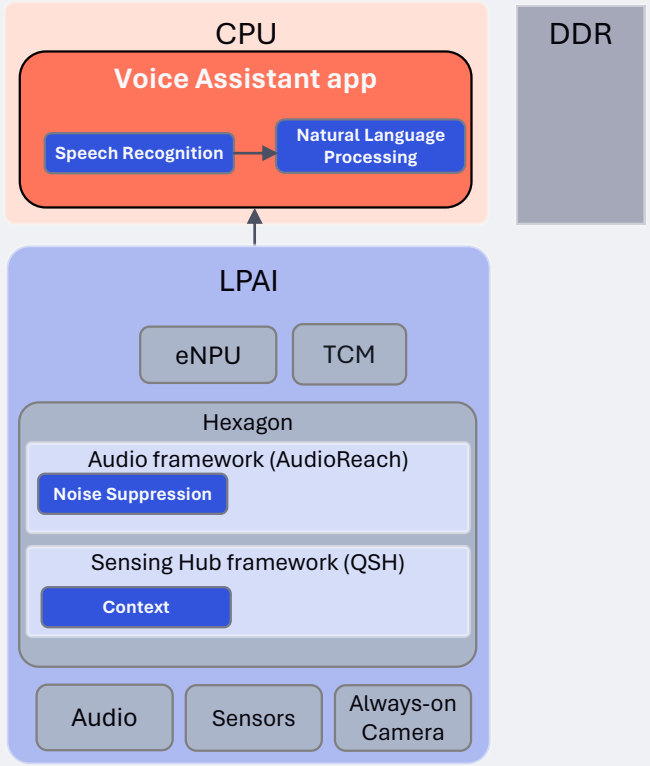


Low Power

Shallow Wake / low power DDR



CPU Active mode



Operating Model	CPU off	CPU off	CPU on
Memory footprint	TCM + LLC*	DDR	DDR
Technologies†	Always-Sensing Camera Audio, Voice AI Sensors, Sensing Hub	Always-Sensing Camera Audio, Voice AI Sensors, Sensing Hub	Always-Sensing Camera Audio, Voice AI Sensors, Sensing Hub
Use Cases	AI runtime (QNN) + SW framework (Audio (AudioReach) or Qualcomm Sensing Hub (QSH))		
Distribution & Security	Signed code by OEM		

Note * Not all chipsets include LLC in LPI.
† LPI technologies very by chipset

LPAI QNN release for SM8750 w/ eNPU v5: Structure

QAISW Release directory structure

```
Stripped/qaisw-v2.xx-xxx/lib
    /hexagon-v79/unsigned
        libQnnLpaiV79Skel_v5.so
    /aarch64-android
        libQnnLpai.so
        libQnnLpaiV79Stub.so
    /x86_64-linux-clang
        libQnnLpai.so
        libQnnLpaiNetRunExtensions.so
        libQnnLpaiPrepare_v5.so
        libQnnLpaiSim_v5.so
```

X86 simulation

libQnnLpai.so (64bit)

Load libQnnLpaiSim_v5.so

ARM Solution

libQnnLpai.so

Load qnnLpaiV79Stub.so

Single QNN Stub library

qnnLpaiV79Stub.so

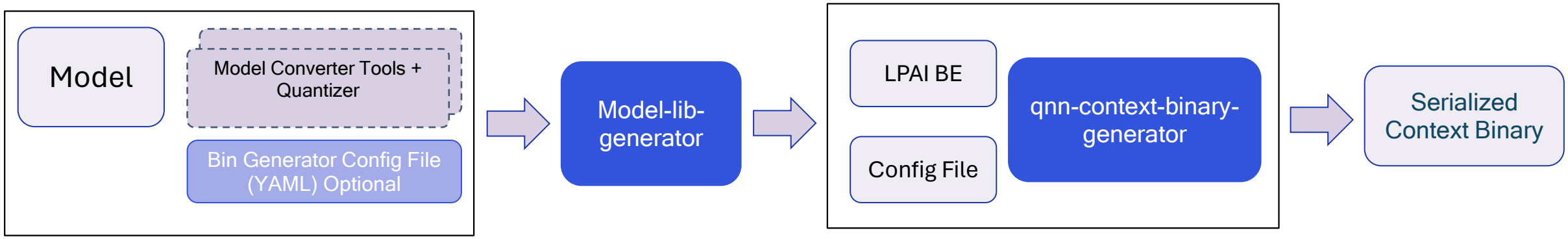
Load libQnnLpaiVXSkel _v5.so

QNN aDSP 32bit Skel library

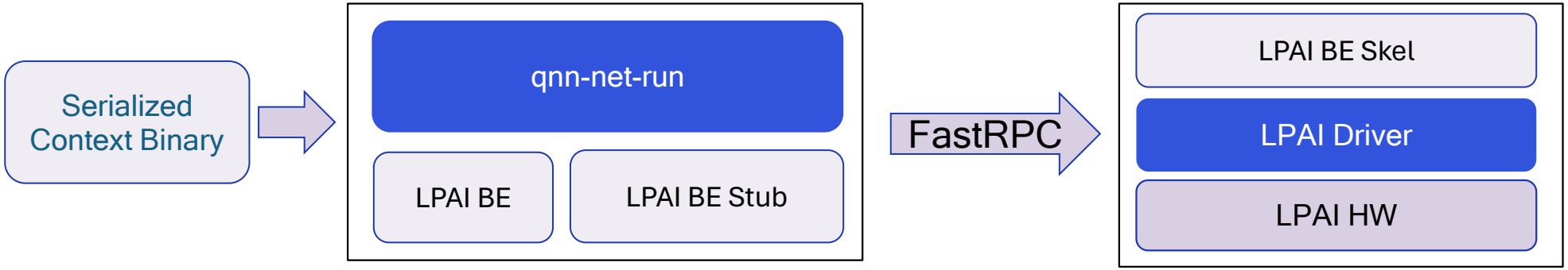
libQnnLpaiV79Skel_v5.so

High-Level end-to-end workflow

Offline



Online



Notes :

1. Offline model preparation is done on x86_64 linux platform
2. Execution is done by aDSP through FastRPC connection

SM8650 Feature Compatibility

With the inclusion of SM8650 skeletons in SDK v2.26.1 release, users can expect enhanced compatibility.

Feature Compatibility Matrix and Known limitations

Features	SM8650 Compatibility	Known Issues
Self-Speculative Decoding (SSD)	✓	
Lookahead Decoding	✓	
Native LoRAv2 Support (LLM)	✓	KI: HTP: LVM+Lora does not work on SM8650.
Block Quantization	x	KI: HTP: Graph compose failure during save context.
Speculative Decoding	✓	
Asynchronous Initialization & Other Improvements	✓	
RAM Optimization: Large Graph Support (>12B)	NA	Not applicable for SM8650
Text to Text Tool Support for HTP	✓	

OEM Application Notes

Network-specific implementation details and best practices for the Neural Processing SDK

Index

Workload Priority Control Support

- Runtime support:
  - CPU ☐, GPU ☐
    DSP ☑, AIP ☑

Workload Priority Control in SNPE

- Provide API to enable OEMs to prioritize DSP & HTA workloads on DSP and AIP runtimes.
- A new API and calling sequence is supported to enable control of workload priorities.
- The control of workload priorities is gated through a SNPE API validated through an OEM supplied shared library
  - The key format and usage are determined by the OEM
- Control has SNPE inference object scope (i.e. per model)

New API (part 1 of 2)
Key configuration

- The API key is configured using the new Snpe_PlatformConfig_SetPlatformOptions() API method in DISystem/PlatformConfig.h

```
// set key determined by OEM
Snpe_PlatformConfig_Handle_t platformConfigHandle = Snpe_PlatformConfig_Create();
Snpe_PlatformConfig_SetPlatformOptions(platformConfigHandle, "oem:<determined-by-OEM>");
```

- Note:“oem” is part of interface specification, and represents the option name
- The call to Snpe_PlatformConfig_SetPlatformOptions() fails if the options string fails validation
- There is a method to validate API key value

```
Snpe_PlatformConfig_IsOptionsValid(platformConfigHandle );
```

- Finally, the handle must be deleted as

```
Snpe_PlatformConfig_Delete(platformConfigHandle);
```

New API (part 2 of 2)

Workload Priority Control

- The control of DSP/HTA workload priority is enabled using the existing Snpe_SNPEBuilder_SetExecutionPriorityHint() API. E.g.

```
// created SNPE instance and set execution hint
Snpe_SNPEBuilder_Handle_t snpeBuilderHandle = Snpe_SNPEBuilder_Create(containerHandle);
Snpe_SNPEBuilder_SetExecutionPriorityHint(snpeBuilderHandle, SNPE_EXECUTION_PRIORITY_HIGH);
…… //Set other SNPEBuilder options
Snpe_SNPE_Handle_t snpeHandle = Snpe_SNPEBuilder_Build(snpeBuilderHandle)
…… //Do Stuff with snpeHandle
Snpe_SNPE_Delete(snpeHandle);
Snpe_SNPEBuilder_Delete(snpeBuilderHandle);
```

- AIP or DSP runtime workload priority setting to HIGH will be allowed only if a valid API key was provided
- Default ExecutionPriorityHint value is NORMAL
- Setting the workload priority to LOW is always allowed

API Validation via OEM Supplied Shared Library

Interface for validating key

- OEM supplies `libsnpe_oem.so` that has a single export

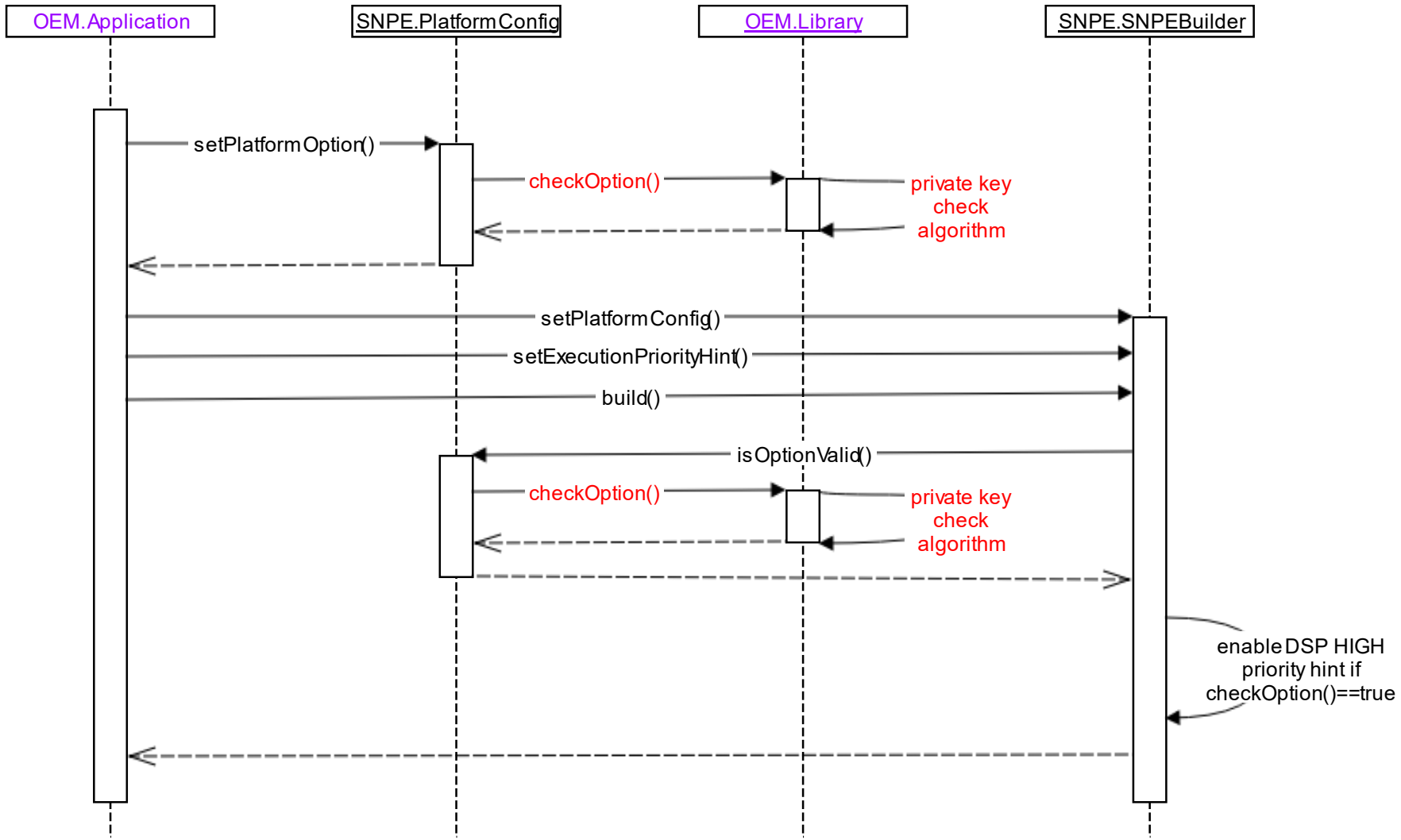
```
extern "C" bool checkOption(const char* optionkey);
```

- Shared library shall be placed in `/system/lib` or `/vendor/lib` depending on the expected usage of SNPE on the OEM platform
- SNPE library will load the `libsnpe_oem.so` library and call the function `checkOption()` to check validity

```
// call function in libsnpe_oem.so for validation
char* option = std::string("<determined-by-OEM>").c_str();
bKeyValid = checkOption(option);
```

- The contents and interpretation of the "oem" option is determined by the OEM customer

Call Sequence for Validation with OEM Shared Library



Sample Implementation of libsnpe_oem.so (part 1 of 2)

Instructions

```
$ mkdir jni
# copy Android.mk, Application.mk and snpe_oem.cpp to jni directory
$ $ANDROID_NDK_ROOT/ndk-build NDK_TOOLCHAIN_VERSION=clang APP_STL=none  # for building with clang
$ adb push libs/armeabi-v7a/libsnpe_oem.so /vendor/lib/
```

Note that the compiler toolchain used for `libsnpe_oem.so` must match the `libSNPE.so` target. Use `NDK_TOOLCHAIN_VERSION=4.9` to build for GCC.

Application.mk

```
# Application.mk
APP_PLATFORM := android-16
APP_ABI := armeabi-v7a arm64-v8a
APP_CPPFLAGS += -std=c++11 -fexceptions -frtti
```

Android.mk

```
# Android.mk
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := libsnpe_oem
LOCAL_SRC_FILES := snpe_oem.cpp
include $(BUILD_SHARED_LIBRARY)
```

Sample Implementation of libsnpe_oem.so (part 2 of 2)

snpe_oem.cpp

```
// snpe_oem.cpp
#include <string.h>

extern "C"{

// Simple key validator. Valid keys have odd number of zeroes
bool checkOption(const char* input_key){
    int zero_count = 0;

    // Check that input_key is not null
    if (input_key != NULL){
        char* keyptr = (char*) input_key;
        int   keylen = strlen(input_key);

        // For each character in input_key, increment zero_count if zero character
        for (int char_checked = 0; char_checked < keylen; char_checked++){
            if (*(keyptr++) == '0')
                zero_count++;
        }
    }

    // Return true if odd number of zeroes
    return (zero_count % 2);
}
}
```

Valid directories for libsnpe_oem.so/libsnpe_oem_32bit.so

- Hexagon/DSP will look for libsnpe_oem_32bit.so/libqaisw_oem_32bit.so
- Android will look for libsnpe_oem.so/libqaisw_oem.so
- Directories in which the search will take place:
  - /system/lib64/
  - /vendor/lib64/
  - /system/lib/
  - /vendor/lib/
  - /lib/
  - /usr/lib/

Sample Implementation of libsnpe_oem_32bit.so for DSP

Instructions

```
$ mkdir jni
# set the required env shown below
# copy Makefile snpe_oem.cpp to jni directory
# make -f Makefile
$ adb push libs/hexagon/libsnpe_oem_32bit.so /vendor/lib/
```

env

```
export HEXAGON_SDK_ROOT=/path/to/hexagon_sdk
export PATH=$HEXAGON_SDK_ROOT/tools/HEXAGON_Tools/8.7.03/Tools/bin:$PATH
```

Makefile

```
HEXAGON_CLANG := $(HEXAGON_SDK_ROOT)/tools/HEXAGON_Tools/8.8.001/Tools/bin/hexagon-clang
HEXAGON_LINK := $(HEXAGON_SDK_ROOT)/tools/HEXAGON_Tools/8.8.001/Tools/bin/hexagon-link
OBJ_DIR := obj/hexagon
LIB_DIR := libs/hexagon
TARGET := $(LIB_DIR)/libsnpe_oem_32bit.so
SOURCES := snpe_oem.cpp
OBJECTS := $(OBJ_DIR)/libsnpe_oem_32bit.o
CFLAGS := -fPIC -mv73
LDFLAGS := -shared -mv73
$(OBJ_DIR)/%.o: %.cpp
        @mkdir -p $(OBJ_DIR)
        $(HEXAGON_CLANG) $(CFLAGS) -c $< -o $@
$(TARGET): $(OBJECTS)
        @mkdir -p $(LIB_DIR)
        $(HEXAGON_CLANG) $(OBJECTS) -o $(TARGET) $(LDFLAGS)
clean:
        rm -f $(OBJECTS) $(TARGET)
```

DSP Memory Grow Size Control Support

- Runtime support:
  - CPU ☐, GPU ☐
    DSP ☑, AIP ☐

DSP Memory Grow Size Control in SNPE

- Extends the existing API to enable user control to set the memory grow size for the DSP runtime
  - The user may set the memory grow size for the DSP runtime by providing the option name and value in the platform configuration options
- Control has SNPE process scope (i.e. per DSP runtime load)
  - The first SNPE inference object using the DSP runtime will set the memory grow size or the default (16 MB) will be used. It is set for the lifetime of the process that uses the DSP runtime. Settings from the second SNPE inference object and using the DSP runtime and later are ignored.

DSP Memory Grow Size Control Support

DSP runtime memory grow size configuration

DSP runtime memory grow size can be configured using the Snpe_PlatformConfig_SetPlatformOptions() API method in DlSystem/PlatformConfig.h

```
// set desired DSP runtime memory grow size
Snpe_PlatformConfig_Handle_t platformConfigHandle = Snpe_PlatformConfig
_Create();
Snpe_PlatformConfig_SetPlatformOptions(platformConfigHandle,
                    "dspGrowSize:<memory-grow-size-value>");
Snpe_PlatformConfig_Delete(platformConfigHandle);
```

- Notes
  - "dspGrowSize" is part of interface specification and represents option name
  - The value must be of base 10 and the supported range is from 0 to 4294967294 (2^32-2 or 0xFFFFFFFE)
  - E.g. "dspGrowSize:33554432" for 32 MB setting

- DSP Runtime memory grow size may be set with other options. Format for using multiple option names is: "optionName1:optionValue1;optionName2:optionValue2", semi-colon is separator

DSP Stub/Skel Loading Behavior (Windows)

Documented in 1.60.0

- When SNPE.dll is loaded, the stub library will be loaded in the following search order:
  1. The directory where SNPE.dll is located.
  2. Standard Search Order for Desktop Applications. (link)

- When stub library is loaded, the skel library will be loaded in the following search order:
  - If DSPRPC_SET_PATH API is used to set the skel path:
    1. The directory where SNPE.dll is located.
    2. The paths set by DSPRPC_SET_PATH API.
  - If DSPRPC_SET_PATH API is not used, SNPE will use the default search paths:
    1. The directory where SNPE.dll is located.
    2. The current directory.
    3. C:\Windows\System32\drivers\DriverData\QUALCOMM\fastRPC.

New Low Level Perf APIs added in SNPE-2.22 (Part 1 of 8)

Highlights

- In SNPE-2.22, we are introducing a set of low level performance management APIs primarily pertaining to the DSP runtime - SNPEPerfProfile.h (C++ Wrapper SNPEPerfProfile.hpp). In addition to this, two new perf profile setting APIs are added to SNPE and one new API to Builder. Details are in Part 3/8.

- These new APIs provide a way for a client application to set different perf settings for initialization, between inferences, and for de-initialization. Up until SNPE-2.21, a user-supplied preset profile (defaulting to Balanced) would be applied during the entire lifecycle of SNPE.

- The new APIs are designed to build a perf profile from the ground up OR start with a preset profile (like Burst) and tweak certain parameters on a case-by-case basis. Such customized profiles (or hybrid profiles) can be applied both during initialization via the new builder API or after initialization for inference and de-initialization using the new SNPE API. Detailed usage examples are in Part 3/8 and 4/8.

- SNPE employs certain heuristics based on the preset profile – which was true even in SNPE-2.21. As part of the new custom perf profile, we have exposed them as knobs that can be tweaked via APIs in SNPEPerfProfile.h. Details are in Part 5/8.

- The new APIs will enable better synchronization flexibility for cooperative multiple SNPE instances within the same process or in different processes pertaining to aggregation of voting packets at the HTP/NSP level. Example with an use case is in Part 6/8 and 7/8.

- The net run applications snpe-net-run and snpe-throughput-net-run have a new command-line argument --perf_config_yaml that takes in a user-provided YAML config file that can configure all the perf settings. A sample YAML config corresponding to sustained high performance profile is provided in Part 8/8.

- PSNPE API configurability options will be available in SNPE-2.23

New Low Level Perf APIs added in SNPE-2.22 (Part 2 of 8)

SNPE-2.21 and prior performance tuning capabilities for DSP

SNPE API

BURST
SUSTAINED_HIGH_PERFORMANCE
HIGH_PERFORMANCE
BALANCED
LOW_BALANCED
HIGH_POWER_SAVER
POWER_SAVER
LOW_POWER_SAVER
EXTREME_POWER_SAVER
SYSTEM_SETTINGS

Current Preset Profiles

Snpe_SNPEBuilder_SetPerformanceProfile()

SNPEBuilder::setPerformanceProfile()

Perf Profile driven heuristics in SNPE

- ✓ RPC Polling is enabled for BURST, SHP and HP (only for Fast RPC based transport and not MCDM based architectures)
- ✓ Voting hysteresis timer of 300 ms for BURST and SHP
- ✓ Async Voting thread for all profiles except BURST and SHP
- ✓ No voting for SYSTEM_SETTINGS as the client is expected to vote directly utilizing Hexagon SDK

BEFORE INIT

AFTER INIT

BEFORE INFERENCE

AFTER INFERENCE

BEFORE DE-INIT

AFTER DE-INIT

- The current performance setting builder API allows the selection of preset profiles during SNPE initialization.
- This preset profile is used for voting both before and after initialization, inference, and de-initialization.
- Based on these preset profiles, SNPE internally employs certain heuristics with respect to RPC polling, async voting thread, and hysteresis timer for voting.
- The limitation of this design is its lack of flexibility. The same profile is used for the entire lifecycle of SNPE. Additionally, the client application is stuck with the corner voltages selected by these preset profiles, even if alternate voltages are offered by the hardware that may better suit the performance-power trade-off for a particular use case.

New Low Level Perf APIs added in SNPE-2.22 (Part 3 of 8)

New perf APIs in SNPE-2.22

- Introducing low level perf API header SNPEPerfProfile.h (C++ Wrapper SNPEPerfProfile.hpp) that allows client application to create a SNPE perf profile handle/object

```
Snpe_SNPEPerfProfile_Handle_t perfHandle = Snpe_SNPEPerfProfile_Create();
SNPEPerfProfile perfProfile; // C++ Wrapper API
```

- A perf profile handle can also be created with a preset perf profile as a starting point

```
Snpe_SNPEPerfProfile_Handle_t perfHandle = Snpe_SNPEPerfProfile_CreatePreset(SNPE_PERFORMANCE_PROFILE_BURST);
SNPEPerfProfile perfProfile(BURST); // C++ Wrapper API
```

- This perf profile handle can now be customized as follows:-

```
Snpe_SNPEPerfProfile_SetDcvsVoltageCornerDcvsVCornerMinDone(perfHandle, SNPE_DCVS_VOLTAGE_VCORNER_NOM_PLUS);
perfProfile.SetDcvsVoltageCornerDcvsVCornerMinDone(DCVS_VOLTAGE_VCORNER_NOM_PLUS); // C++ Wrapper API
```

- This custom perf profile handle/object can now be associated with SNPE Builder handle (for init) or SNPE handle (for inference and de-init)

```
Snpe_SNPEBuilder_SetCustomPerfProfile(snpeBuilderHandle, perfHandle);
builder.setCustomPerfProfile(perfProfile); // C++ Wrapper API
Snpe_SNPE_SetCustomPerfProfile(snpeHandle, perfHandle);
snpe->setCustomPerformanceProfile(perfProfile); // C++ Wrapper API
Snpe_SNPE_ExecuteUserBuffers(snpeHandle, inputHandle, outputHandle);
snpe->execute(inputUbMap, outputUbMap); // C++ Wrapper API
```

App Note D

New Low Level Perf APIs added in SNPE-2.22 (Part 4 of 8)

New perf APIs in SNPE-2.22

- An example usage with C API showing different perf settings for init, between inferences and for de-init

```
Snpe_SNPEPerfProfile_Handle_t perfHandle = Snpe_SNPEPerfProfile_CreatePreset(SNPE_PERFORMANCE_PROFILE_HIGH_PERFORMANCE);
Snpe_SNPEPerfProfile_SetCoreVoltageCornerMaxMvStart(perfHandle, SNPE_DCVS_VOLTAGE_VCORNER_TURBO_PLUS);
Snpe_SNPEPerfProfile_SetCoreVoltageCornerMaxMvDone(perfHandle, SNPE_DCVS_VOLTAGE_VCORNER_SVS);
Snpe_SNPEBuilder_SetCustomPerfProfile(snpeBuilderHandle, perfHandle); // perf settings set to high_performance + new corners
Snpe_SNPEPerfProfile_Delete(perfHandle);
Snpe_SNPE_Handle_t snpeHandle = Snpe_SNPEBuilder_Build(snpeBuilderHandle); //Init performed with this custom setting
Snpe_SNPE_ExecuteUserBuffers(snpeHandle, inputHandle, outputHandle); // 1st inference with this custom setting

Snpe_SNPE_SetPerformanceProfile(snpeHandle, SNPE_PERFORMANCE_PROFILE_BURST); //perf setting now overwritten to Burst
Snpe_SNPE_ExecuteUserBuffers(snpeHandle, inputHandle, outputHandle); // 2nd inference with preset Burst profile

Snpe_SNPEPerfProfile_Handle_t perfHandle2 = Snpe_SNPEPerfProfile_Create();
Snpe_SNPEPerfProfile_SetBusVoltageCornerMinStart(perfHandle2, SNPE_DCVS_VOLTAGE_VCORNER_NOM);
…… more settings ……
Snpe_SNPE_SetCustomPerfProfile(snpeHandle, perfHandle2); // perf settings for the SNPE handle set to cutom values
Snpe_SNPEPerfProfile_Delete(perfHandle2);
Snpe_SNPE_ExecuteUserBuffers(snpeHandle, inputHandle, outputHandle); // 3rd inference with custom perf settings

Snpe_SNPE_SetPerformanceProfile(snpeHandle, SNPE_PERFORMANCE_PROFILE_LOW_BALANCED); //perf setting now overwritten to preset
Snpe_SNPE_Delete(snpeHandle); //De-Init performed with this new preset of low_balanced profile
```

New Low Level Perf APIs added in SNPE-2.22 (Part 5 of 8)

Ability to hold the vote for a longer duration – minimizing RPC calls

With the preset profiles burst and sustained high performance, inference voting has a hysteresis feature. Once clocks are bumped up before execute, they are not brought down after inference is done for a certain period (300 ms) to reduce the amount of voting packets sent to the NSP. This feature is present in SNPE-2.21 and particularly helps when there are back to back inferences. With SNPE-2.22, we are exposing this as part of custom perf profile. So the hysteresis timer can be increased or reduced or totally turned off for burst and sustained high performance. Also this can now be enabled for any other perf profile. Example:-

```
Snpe_SNPEPerfProfile_Handle_t perfHandle = Snpe_SNPEPerfProfile_CreatePreset(SNPE_PERFORMANCE_PROFILE_LOW_POWER_SAVER);
Snpe_SNPEPerfProfile_SetDspHysteresisTime(perfHandle, 500); //Set inference voting hysteresis to 500 ms
Snpe_SNPE_SetCustomPerfProfile(snpeHandle, perfHandle);
Snpe_SNPE_ExecuteUserBuffers(snpeHandle, inputHandle, outputHandle);

…… more inferences ……

Snpe_SNPEPerfProfile_SetDspHysteresisTime(perfHandle, 0); //Disable voting hysteresis after a few inferences
Snpe_SNPE_SetCustomPerfProfile(snpeHandle, perfHandle);
Snpe_SNPE_ExecuteUserBuffers(snpeHandle, inputHandle, outputHandle);
```

New Low Level Perf APIs added in SNPE-2.22 (Part 6 of 8)

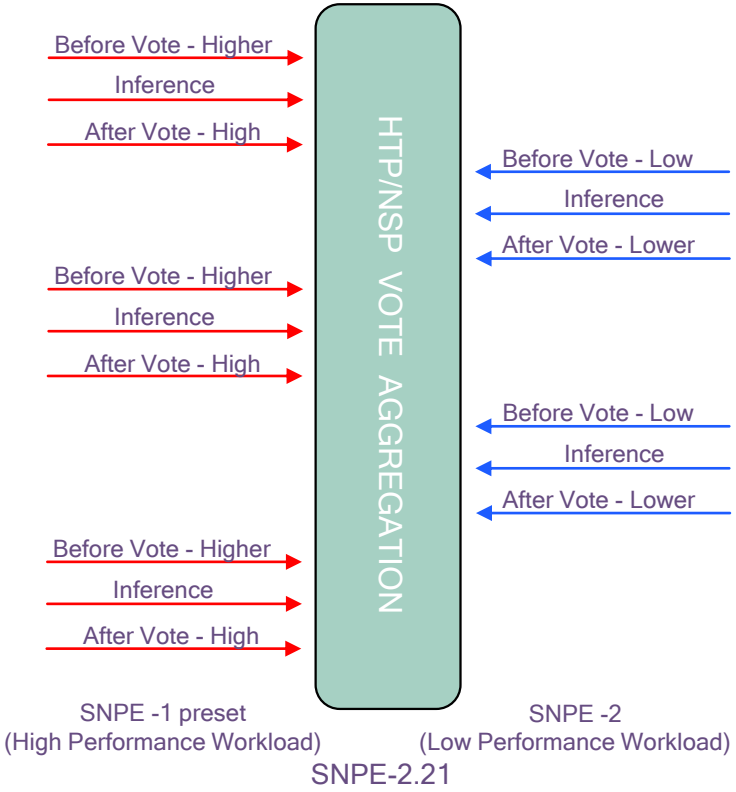
Synchronization problem for multiple SNPE instances in SNPE-2.21

Consider the use case of two SNPE instances, as shown in the diagram on the right. These instances could be part of the same process or different processes. They are executing concurrently, with one having a high-performance workload (represented in red) and the other having a low-performance workload (represented in blue).

When these two SNPE instances are in separate processes, they get unique power config ids. When they are in the same process and unique power config ids were requested via Platform Option “dspPowerSettingContext” (which is the default), they get different power config ids.

Due to this, during vote aggregation at the HTP/NSP level, the after/done vote of SNPE-1 would vote out the before/start vote of SNPE-2. This is because the default after/done vote of SNPE-1 is still high compared to the before/start vote of SNPE-2. The before and after voting voltage corners are controlled by a preset performance profile that was passed during builder/init stage. This means the execution of SNPE-2 would be performed at a higher performance level, which is not intended and thus ends up consuming more power.

With the introduction of the new perf APIs in SNPE-2.22, this can be solved which is outlined in the next slide.

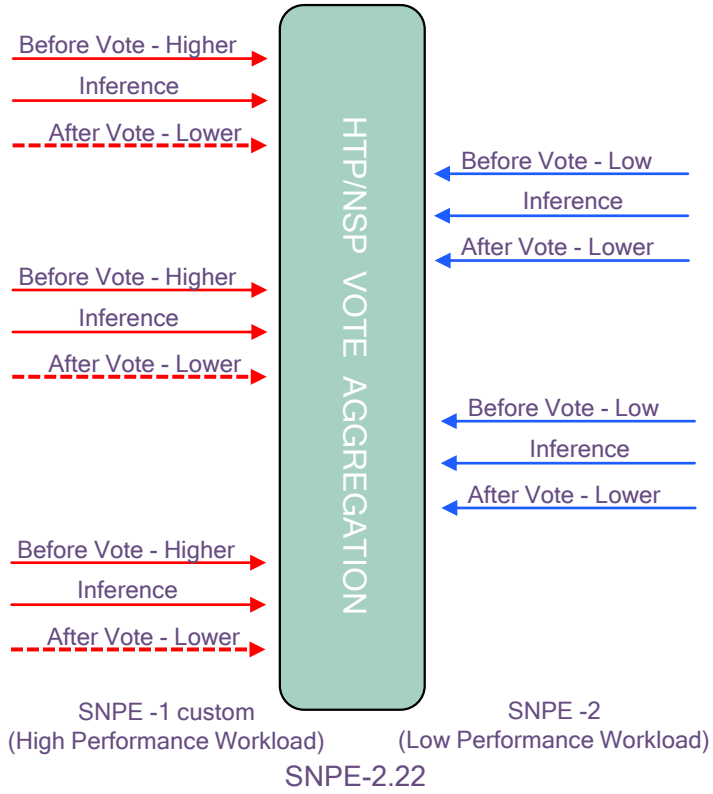


New Low Level Perf APIs added in SNPE-2.22 (Part 7 of 8)

Better synchronization flexibility for cooperative multiple SNPE instances

With the introduction of the new performance APIs (illustrated in the diagram on the right), SNPE-1 can now selectively control the after/done vote (indicated by the dotted red arrow). SNPE-1 can match the after/done vote voltage corner to match SNPE-2's before/start vote or lower than that or not vote at all. This allows it to cooperate with SNPE-2, thus enabling SNPE-2 to execute in desired low power mode. Code snippet to lower the after/done vote for SNPE-1 the high performance workload is provided below:-

```
// C API
Snpe_SNPEPerfProfile_Handle_t perfHandle1 = Snpe_SNPEPerfProfile_CreatePreset(SNPE_PERFORMANCE_PROFILE_BURST);

Snpe_SNPEPerfProfile_SetBusVoltageCornerMinDone(perfHandle1, SNPE_DCVS_VOLTAGE_CORNER_DISABLE);
Snpe_SNPEPerfProfile_SetBusVoltageCornerTargetDone(perfHandle1, SNPE_DCVS_VOLTAGE_CORNER_DISABLE);
Snpe_SNPEPerfProfile_SetBusVoltageCornerMaxDone(perfHandle1, SNPE_DCVS_VOLTAGE_CORNER_DISABLE);

Snpe_SNPEPerfProfile_SetCoreVoltageCornerMinMvDone(perfHandle1, SNPE_DCVS_VOLTAGE_CORNER_DISABLE);
Snpe_SNPEPerfProfile_SetCoreVoltageCornerTargetMvDone(perfHandle1, SNPE_DCVS_VOLTAGE_CORNER_DISABLE);
Snpe_SNPEPerfProfile_SetCoreVoltageCornerMaxMvStart(perfHandle1, SNPE_DCVS_VOLTAGE_CORNER_DISABLE);

Snpe_SNPE_SetCustomPerfProfile(snpe1Handle, perfHandle1);
Snpe_SNPE_ExecuteUserBuffers(snpeHandle1, inputHandle, outputHandle);


// C++ Wrapper API
SNPEPerfProfile perfProfile1(BURST);
perfProfile1.setBusVoltageCornerMinDone(DCVS_VOLTAGE_CORNER_DISABLE);
perfProfile1.setBusVoltageCornerTargetDone(DCVS_VOLTAGE_CORNER_DISABLE);
perfProfile1.setBusVoltageCornerMaxDone(DCVS_VOLTAGE_CORNER_DISABLE);

perfProfile1.setCoreVoltageCornerMinMvDone(DCVS_VOLTAGE_CORNER_DISABLE);
perfProfile1.setCoreVoltageCornerTargetMvDone(DCVS_VOLTAGE_CORNER_DISABLE);
perfProfile1.setCoreVoltageCornerMaxMvDone(DCVS_VOLTAGE_CORNER_DISABLE);
snpe1->setCustomPerformanceProfile(perfProfile1);
snpe1->execute(inputUbMap, outputUbMap);
```



App Note D

New Low Level Perf APIs added in SNPE-2.22 (Part 8 of 8)

snpe-net-run yaml corresponding to SUSTAINED_HIGH_PERFORMANCE profile

```
general:
  ASYNC_VOTING_ENABLE: false
  DSP_HYSTERESIS_TIME_US: 300000
  DSP_SLEEP_DISABLE_MS: 0
  DSP_RPC_POLLING_TIME_US: 9999
init:
  DSP_ENABLE_DCVS_START: false
  DSP_ENABLE_DCVS_DONE: true
  DSP_SLEEP_LATENCY_START_US: 100
  HIGH_PERFORMANCE_MODE: true
  POWERMODE_START: SNPE_DSP_PERF_INFRASTRUCTURE_POWERMODE_ADJUST_UP_DOWN
  POWERMODE_DONE: SNPE_DSP_PERF_INFRASTRUCTURE_POWERMODE_ADJUST_UP_DOWN
  BUS_VOLTAGE_CORNER_MIN_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_TARGET_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_MAX_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_MIN_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_TARGET_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_MAX_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_MIN_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS2
  BUS_VOLTAGE_CORNER_TARGET_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS
  BUS_VOLTAGE_CORNER_MAX_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS
  CORE_VOLTAGE_CORNER_MIN_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS2
  CORE_VOLTAGE_CORNER_TARGET_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS
  CORE_VOLTAGE_CORNER_MAX_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS
  DSP_SLEEP_LATENCY_DONE_US: 2000
execute:
  DSP_ENABLE_DCVS_START: false
  DSP_ENABLE_DCVS_DONE: true
  DSP_SLEEP_LATENCY_START_US: 100
  HIGH_PERFORMANCE_MODE: true
  POWERMODE_START: SNPE_DSP_PERF_INFRASTRUCTURE_POWERMODE_ADJUST_UP_DOWN
  POWERMODE_DONE: SNPE_DSP_PERF_INFRASTRUCTURE_POWERMODE_ADJUST_UP_DOWN
  BUS_VOLTAGE_CORNER_MIN_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_TARGET_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_MAX_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_MIN_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_TARGET_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_MAX_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_MIN_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS2
  BUS_VOLTAGE_CORNER_TARGET_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS
  BUS_VOLTAGE_CORNER_MAX_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS
  CORE_VOLTAGE_CORNER_MIN_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS2
  CORE_VOLTAGE_CORNER_TARGET_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS
  CORE_VOLTAGE_CORNER_MAX_DONE: SNPE_DCVS_VOLTAGE_VCORNER_SVS
  DSP_SLEEP_LATENCY_DONE_US: 2000
continued to right..
```

```
… continued from left

deinit:
  DSP_ENABLE_DCVS_START: false
  DSP_ENABLE_DCVS_DONE: true
  HIGH_PERFORMANCE_MODE: true
  DSP_SLEEP_LATENCY_START_US: 100
  POWERMODE_START: SNPE_DSP_PERF_INFRASTRUCTURE_POWERMODE_ADJUST_UP_DOWN
  POWERMODE_DONE: SNPE_DSP_PERF_INFRASTRUCTURE_POWERMODE_ADJUST_UP_DOWN
  BUS_VOLTAGE_CORNER_MIN_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_TARGET_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_MAX_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_MIN_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_TARGET_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  CORE_VOLTAGE_CORNER_MAX_START: SNPE_DCVS_VOLTAGE_VCORNER_TURBO
  BUS_VOLTAGE_CORNER_MIN_DONE: SNPE_DCVS_VOLTAGE_VCORNER_MIN_VOLTAGE_CORNER
  BUS_VOLTAGE_CORNER_TARGET_DONE: SNPE_DCVS_VOLTAGE_VCORNER_MIN_VOLTAGE_CORNER
  BUS_VOLTAGE_CORNER_MAX_DONE: SNPE_DCVS_VOLTAGE_VCORNER_MIN_VOLTAGE_CORNER
  CORE_VOLTAGE_CORNER_MIN_DONE: SNPE_DCVS_VOLTAGE_VCORNER_MIN_VOLTAGE_CORNER
  CORE_VOLTAGE_CORNER_TARGET_DONE: SNPE_DCVS_VOLTAGE_VCORNER_MIN_VOLTAGE_CORNER
  CORE_VOLTAGE_CORNER_MAX_DONE: SNPE_DCVS_VOLTAGE_VCORNER_MIN_VOLTAGE_CORNER
  DSP_SLEEP_LATENCY_DONE_US: 2000
```

Note:- These settings can be retrieved via the getter API s after starting with a preset profile. Example-

CAPI

```
Snpe_SNPEPerfProfile_Handle_t perfHandle = Snpe_SNPEPerfProfile_CreatePreset(SNPE_PERFORMANCE_PROFILE_BURST);
auto asyncVoteStatus = Snpe_SNPEPerfProfile_GetEnableAsyncVoting(perfHandle);
auto sleepLatencyStart = Snpe_SNPEPerfProfile_GetSleepLatencyStart(perfHandle);
auto busVoltageCornerMinDone = Snpe_SNPEPerfProfile_GetBusVoltageCornerMinDone(perfHandle);
……
……
```

C++ Wrapper API

```
SNPEPerfProfile perfProfile(BURST);
auto asyncVoteStatus = perfProfile.getEnableAsyncVoting();
auto sleepLatencyStart = perfProfile.getSleepLatencyStart();
auto busVoltageCornerMinDone = perfProfile.getBusVoltageCornerMinDone();
……
……
```

User Level Python API

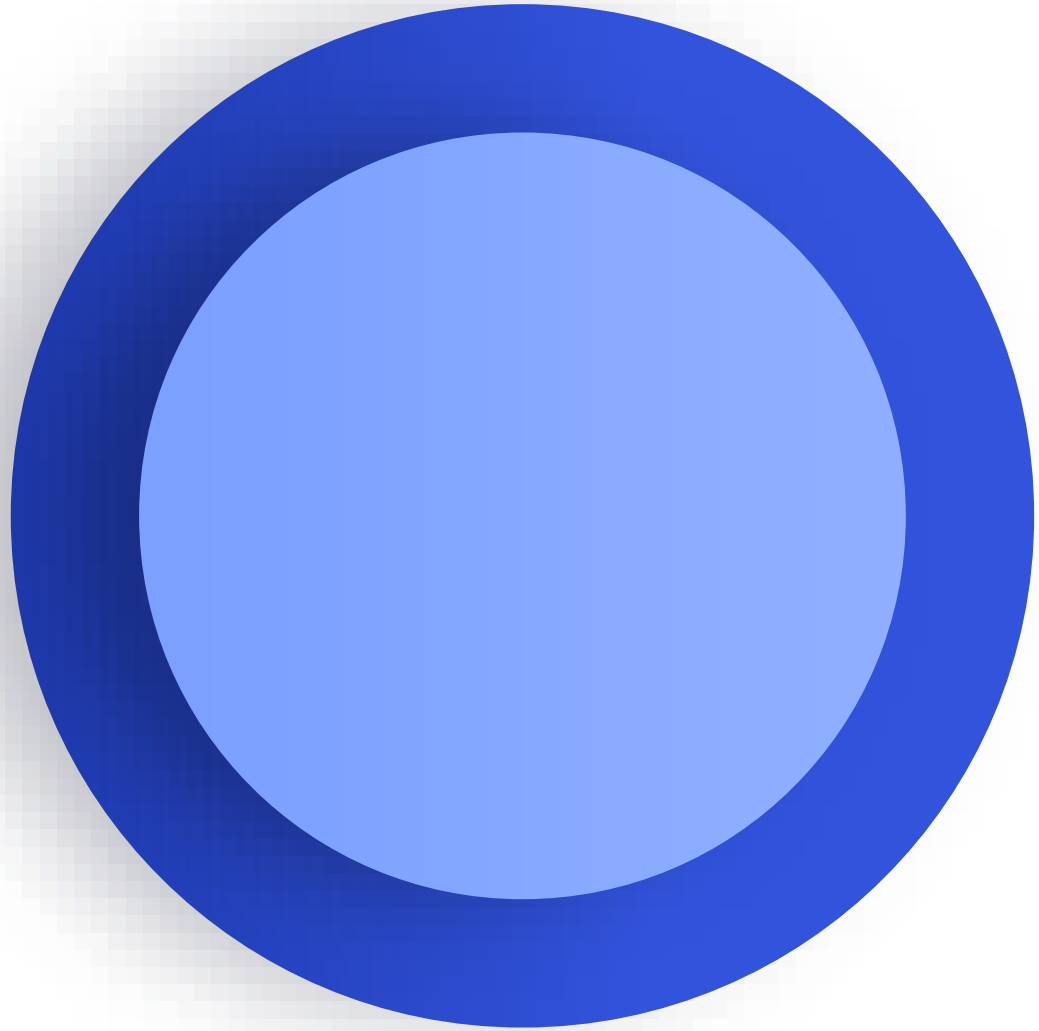
- The User Level Python API offers a simple alternative to QAIRT CLI Tooling, enabling users to convert, compile and execute QAIRT models natively in Python. The User Level Python API replaces the High-Level Python APIs which were added in 2.26.0.
- This is an alpha version that supports:
  - Conversion of ONNX and TFLite Framework models
  - Compilation and Execution
  - Profiling: Op trace and QHAS report generation
  - DLC Editing and Context Binary inspection
- Supported capabilities:
  - OS: Linux, Android (for execution)
  - Model frameworks: ONNX
  - Backends: CPU, GPU, HTP
  - Platforms: x86_64-linux-clang, aarch64-android, aarch64-oe-linux-gcc variants (per QAIRT SDK)
- Known limitations:
  - Support has been verified for ONNX and TFLite models only.
  - Limited support for conversion with encodings (data free)
  - No calibration support

User Level Python API Preview

```
from pathlib import Path

import numpy as np
import qairt

from qairt import CompileConfig
from qairt.api.configs import Device, DevicePlatformType, RemoteDeviceIdentifier

resnet_model = "./resnet50.onnx" # change to your desired path

# Convert the model
converted_model = qairt.convert(resnet_model)

# Compile with default config
compiled_model = qairt.compile(converted_model, backend="HTP")

INPUT_DATA = np.random.rand(1, 224, 224, 3).astype(np.float32)

# Execute the compiled model locally on x86-linux, backend = HTP
exec_result_local = compiled_model(inputs=INPUT_DATA)

# ---------------- Execute the Model on an android device ------------- #
SERIAL_ID = "f609a332"  # retrieve adb serial

device_identifier = RemoteDeviceIdentifier(serial_id=SERIAL_ID)
target_device = Device(identifier=device_identifier, type=DevicePlatformType.ANDROID)

# Execute the compiled model on an android device
exec_result_android = compiled_model(inputs=INPUT_DATA, device=target_device)

# print data
print(exec_result_local.data)

# Save compiled model as context binary file (.bin)
output_dir = Path("./output").mkdir(exist_ok=True)
compiled_model.save(output_dir / "compiled_model.bin")
```

- Native python tooling workflow via simplified APIs
- Enables easy prototyping on host via **pybind libraries** including native numpy support
- Enables target execution on ADB; support for QNX will be added in an upcoming release
- Enables generation of op trace and profiling reports (not pictured)

LLaMA2-7B Model Enablement for QAIRT GPU



Introduction

Default Hugging Face Model Limitations for QNN GPU Enablement

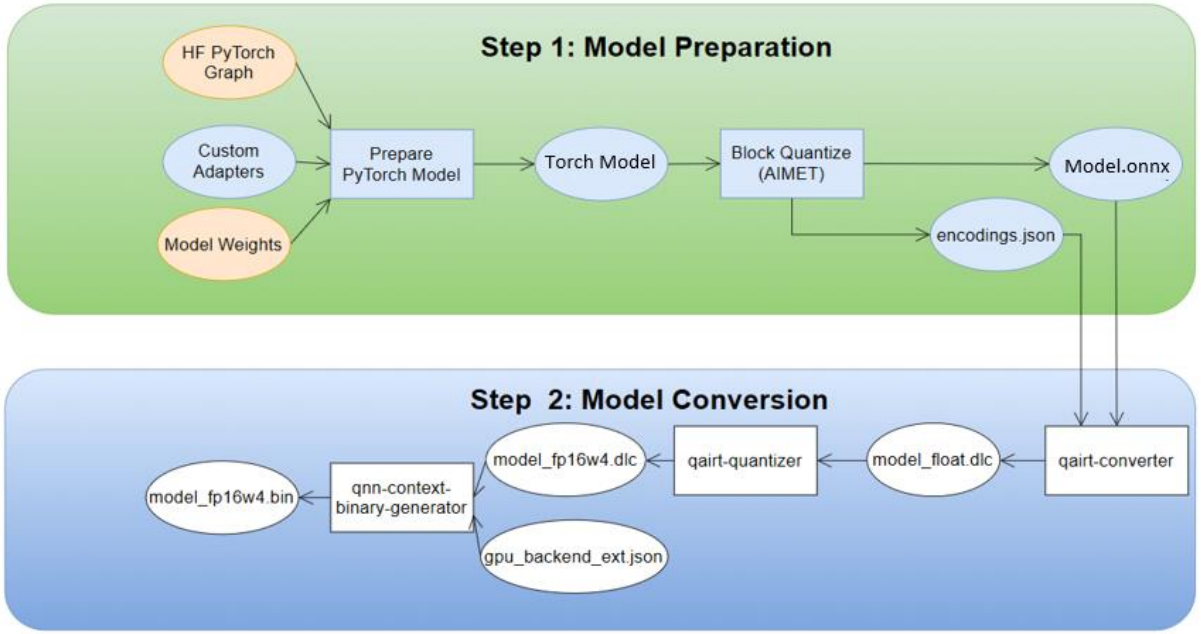
- The Hugging Face model (modeling_llama.py) maintains an internal ‘past_key_values’ buffers, which doubles the memory requirement.
- The Hugging Face model (modeling_llama.py) uses concat to append ‘new_key_value’ to ‘past_key_values’. QNN GPU Backend does not support Dynamic Tensors, so, concat will not work efficiently.

- Requirements
  - Ubuntu 22.04
  - AIMET Pro 1.34.0
  - Transformers (4.41.2)
  - Tokenizers (0.19.0)
  - QAIRT 2.29 SDK
  - SM8750

- Step1: Model preparation
  - Apply custom adapters to the default pytorch graph and generate llama2-7b onnx model and weight encodings (using AIMET Block Quantization)
  - Custom Adaptors
    - Scatter based KV Cache update
    - RoPE G2G update and pre-computed cos/sin embeddings
    - Transpose Key Cache

- Step2: Model conversion
  - qairt-converter : Convert onnx model to FP32 DLC
  - qairt-quantizer : Quantize weights into packed INT4 to generate afp16w4 DLC.
  - qnn-context-binary-generator : Generate serialized context binary from DLC using “TuningApp”. Refer to TuningApp tutorial in QAIRT SDK

- Step3: Model execution
  - Load context binary and execute on SM8750 target.



Results on SM8750 (QAIRT 2.30.0 SDK)

Prompt processing	Token Generation
12.56 tok/s	12.85 tok/s

Step 1 – Model Preparation (Model Adapters)

Adapter 1 : LLaMa Rotary Embeddings

- Store pre-computed cos/sin embedding in the model
- The adapters for Embedding and RoPE are defined below.

```python
class LlamaRotaryEmbedding(torch.nn.Module):
    def __init__(self, dim, max_position_embeddings=2048, base=10000, device=None):
        super().__init__()
        inv_freq = 1.0 / (base ** (torch.arange(0, dim, 2).float().to(device) / dim))
        self.register_buffer("inv_freq", inv_freq)

        # Build here to make `torch.jit.trace` work.
        self.max_seq_len_cached = max_position_embeddings
        t = torch.arange(self.max_seq_len_cached, device=self.inv_freq.device,
                                 dtype=self.inv_freq.dtype)
        freqs = torch.einsum("i,j->ij", t, self.inv_freq)

        emb = torch.cat((freqs, freqs), dim=-1)
        self.register_buffer("cos_cached", emb.cos()[None, None, :, :], persistent=False)
        self.register_buffer("sin_cached", emb.sin()[None, None, :, :], persistent=False)

    def forward(self, x, position_ids=None):
        return (
            self.cos_cached[:, :, :, ...].to(dtype=x.dtype),
            self.sin_cached[:, :, :, ...].to(dtype=x.dtype),
        )
```

```python
def rotate_half(x):
    """Rotates half the hidden dims of the input."""
    x1 = x[..., : x.shape[-1] // 2]
    x2 = x[..., x.shape[-1] // 2 :]
    return torch.cat((-x2, x1), dim=-1)

def _apply_rotary_pos_emb(q, k, cos, sin, position_ids):
    # The first two dimensions of cos and sin are always 1, so we can `squeeze` them.
    cos = cos[0,0,:,:]  # [seq_len, dim]
    sin = sin[0,0,:,:]  # [seq_len, dim]
    cos = cos[position_ids].unsqueeze(1)  # [bs, 1, seq_len, dim]
    sin = sin[position_ids].unsqueeze(1)  # [bs, 1, seq_len, dim]
    q_embed = (q * cos) + (rotate_half(q) * sin)
    k_embed = (k * cos) + (rotate_half(k) * sin)
    return q_embed, k_embed
```

Step 1 – Model Preparation (Model Adapters)

Adapter 2 : LLaMaAttention forward()

- Scatter based KV Cache Update : To optimize memory allocation and use KVCache without Dynamic Tensors. Use pre-defined KVCache for input and output update through Scatter Operation.
- Transpose Key Cache : Allocate the KeyCache with a transposed dimension initially to avoid the costly transpose operation during inference.
- The forward function in LLaMAttention () is customized with the QcLLaMAAttention_forward

QcLLaMaAttention_forward()

```
transposed_key_cache = True
# Custom LlamaAttention Changes
cos, sin = self.rotary_emb(value_states, position_ids)
query_states, key_states = _apply_rotary_pos_emb(
                                query_states, key_states, cos, sin, position_ids)

# After application of RoPE,
# apply transpose on current key_state to match the dimension of key_cache
if transposed_key_cache: # QC
    key_states = key_states.transpose(2, 3)


# Update Key Value Cache with current Key and Value using Scatter
if use_scatter_for_kvcache_update:
    key_data = past_key_value[0]
    value_data = past_key_value[1]
    indexKey = torch.ones(key_states.shape, dtype=int)*position_ids[0]
    indexVal = torch.ones(value_states.shape, dtype=int)*position_ids[0]
    dimKey = 3 if transposed_key_cache else 2
    dimVal = 2
    key_data.scatter_(dimKey, indexKey, key_states)
    value_data.scatter_(dimVal, indexVal, value_states)
    key_states = key_data
    value_states = value_data



# As key_cache is already tranposed, directly matmul with query_states
if transposed_key_cache:
    attn_weights = torch.matmul(query_states, key_states) / math.sqrt(self.head_dim)
else:
    attn_weights = torch.matmul(query_states, key_states.transpose(2, 3)) /
                                math.sqrt(self.head_dim)
```

Step 1 – Model Preparation (Model Adapters)

Adapter 3 : Update Causal Mask

- The default implementation of update_causal_mask() requires cache_position parameter, which conflicts during onnx graph trace.
- To avoid that requirement, overwrite the _update_causal_mask with QcLlamaModel_update_causal_mask.

QcLlamaModel_update_causal_mask()

```python
def QcLlamaModel_update_causal_mask(
    attention_mask: torch.Tensor,
    input_tensor: torch.Tensor,
    past_key_values: Cache,
    mask_neg
):
    bsz, src_len = attention_mask.size()
    dtype = input_tensor.dtype
    tgt_len = src_len
    if past_key_values is not None:
        tgt_len = input_tensor.shape[-2]

    expanded_mask = attention_mask[:, None, None, :].expand(bsz, 1, tgt_len, src_len).to(dtype)
    inverted_mask = 1.0 - expanded_mask
    return inverted_mask.masked_fill(inverted_mask.to(torch.bool), mask_neg)
```

Step 1 – Model Preparation (Model Adapters)

Adapter 4 : Update LLaMaModel forward()

- The HF (modeling_llama.py) LLaMaModel forward() function prepares an additional copy of past_key_values.
- To avoid an additional copy, overwrite the forward() method with custom QcLlamaModel_iocache_forward() implementation

```python
def QcLlamaModel_iocache_forward(
    self,
    input_ids: torch.LongTensor = None,
    attention_mask: Optional[torch.Tensor] = None,
    position_ids: Optional[torch.LongTensor] = None,
    past_key_values: Optional[Union[Cache, List[torch.FloatTensor]]] = None,
    inputs_embeds: Optional[torch.FloatTensor] = None,
    use_cache: Optional[bool] = None,
    output_attentions: Optional[bool] = None,
    output_hidden_states: Optional[bool] = None,
    return_dict: Optional[bool] = None,
    cache_position: Optional[torch.LongTensor] = None,
) -> Union[Tuple, BaseModelOutputWithPast]:

    output_attentions = output_attentions if output_attentions is not None else
                                self.config.output_attentions
    output_hidden_states = (
        output_hidden_states if output_hidden_states is not None else
                self.config.output_hidden_states)

    use_cache = use_cache if use_cache is not None else self.config.use_cache
    return_dict = return_dict if return_dict is not None else self.config.use_return_dict

    use_cache = False
    if inputs_embeds is None:
        inputs_embeds = self.embed_tokens(input_ids)

    mask_neg = self.config.mask_neg if hasattr(self.config, 'mask_neg') else -100
    causal_mask = QcLlamaModel_iocache_update_causal_mask(
        attention_mask, inputs_embeds, past_key_values, mask_neg
    )
```

```python
    # embed positions
    hidden_states = inputs_embeds

    # decoder layers
    all_hidden_states = () if output_hidden_states else None
    all_self_attns = () if output_attentions else None
    next_decoder_cache = None

    for idx, decoder_layer in enumerate(self.layers):
        layer_outputs = decoder_layer(
                hidden_states,
                attention_mask=causal_mask,
                position_ids=position_ids,
                past_key_value=past_key_values[idx],
                output_attentions=output_attentions,
                use_cache=use_cache,
                cache_position=cache_position,
            )

        hidden_states = layer_outputs[0]
    hidden_states = self.norm(hidden_states)
    return BaseModelOutputWithPast(
        last_hidden_state=hidden_states,
        past_key_values=past_key_values,
        hidden_states=all_hidden_states,
        attentions=all_self_attns,
    )
```

Step 1 – Model Preparation (Model Adapters)

Prepare PyTorch Model

- Overwrite classes to use adapter classes
- Use python setattr(cls, attr_name, new_attr) method to apply adapter methods

1. Load default hf (modeling_llama.py)

```
from transformers.models.llama import modeling_llama
```

2. Load Adapters

```
from qcllama_adapters import (
    QcLlamaAttention_forward,
    QcLlamaModel_forward,
    QcLlamaModel_update_causal_mask,
    LlamaRotaryEmbedding
)
```

3. Apply Adapters

```
setattr(modeling_llama.LlamaAttention, 'forward', QcLlamaAttention_forward)
setattr(modeling_llama.LlamaModel,     'forward', QcLlamaModel_forward)
setattr(modeling_llama.LlamaModel,     '_update_causal_mask’, QcLlamaModel_update_causal_mask)

modeling_llama.LlamaRotaryEmbedding = LlamaRotaryEmbedding
```

4. Create Model

```
model = modeling_llama.LlamaForCausalLM.from_pretrained(model_id, config=llm_config)
```

5. Prepare Dummy Inputs

```
If using transposed_key_cache, create key_cache buffers of dimension [1, 32, 128, ctx_len] instead of [1, 32, ctx_len, 128]
```

6. At this stage, we have model with custom graph structure and model weights.

```
a. We can directly use this pytorch model to BlockQuantize FullyConnected Weights (AIMET).
b. We can export this model to onnx using torch.onnx.export and apply other quantization tools. Use onnx-simplifier for optimizing the model further.
```

Note: These adaptations can be applied to other LLMs to generate onnx model that is optimal for execution on QNN GPU.

Step 1 – Model Preparation (Block Quantization)

Block Quantization using AIMET

- Steps to Block Quantize Onnx Model using AIMET with block size = 32. Quantize FullyConnected weights to 4-bit value and generate encoding values.

1. Load the pytorch model

```
from transformers.models.llama import modeling_llama
```

2. Prepare model using AIMET model preparer pro, In this step Linear(QKV) optimization is done through onnx-simplifier.

```
prepared_model = model_preparer.prepare_model(model, dummy_input, filename = {model_name}.onnx,  path=output_dir,
input_names=input_names,                                                    output_names=output_names)
```

3. Generate QuantizationSimModel from the prepared model.

```
quantsim = QuantizationSimModel(model=prepared_model, quant_scheme='tf', dummy_input=dummy_input_tuple,
                                default_output_bw=16, default_param_bw=16, default_data_type=QuantizationDataType.float)
```

4. Compute encodings for Linear Layer

```
for name, module in quantsim.model.named_modules():
    if isinstance(module, QuantizedLinear):
        weight_shape = module.weight.shape
        num_blocks = weight_shape[1] // 32
        module.param_quantizers['weight'] = GroupedBlockQuantizeDequantize(
             shape=(weight_shape[0] , num_blocks), bitwidth=4, symmetric=True,
             decompressed_bw=BITWIDTH, block_size=(1, 32),
        block_grouping=(1, 1))
    module._remove_activation_quantizers()
    if isinstance(module, BaseQuantizationMixin):
        module._remove_activation_quantizers()
```

5. Export Prepared ONNX and Encodings

```
from aimet_torch import onnx_utils
quantsim.encoding_version = '1.0.0'
onnx_api_args = onnx_utils.OnnxExportApiArgs(input_names=input_names,output_names=output_names, opset_version=11)
quantsim.export(onnx_dir, 'llama_v2_bq', dummy_inputs, onnx_export_args=onnx_api_args)
```

Step2 – Model Conversion

1. **Conversion (using qairt-converter)**

```
qairt-converter  --input_network model.onnx  --quantization_overrides encodings.json--output_path model_float.dlc
                 --source_model_input_layout "past_key_0_in" NHWC --source_model_input_layout "past_value_0_in" NHWC
                 --source_model_input_layout "past_key_1_in" NHWC --source_model_input_layout "past_value_1_in" NHWC
                 --source_model_input_layout "past_key_2_in" NHWC --source_model_input_layout "past_value_2_in" NHWC
                 --source_model_input_layout "past_key_3_in" NHWC --source_model_input_layout "past_value_3_in" NHWC
                 --source_model_input_layout "past_key_4_in" NHWC --source_model_input_layout "past_value_4_in" NHWC
                 --source_model_input_layout "past_key_5_in" NHWC --source_model_input_layout "past_value_5_in" NHWC
                 --source_model_input_layout "past_key_6_in" NHWC --source_model_input_layout "past_value_6_in" NHWC
                 --source_model_input_layout "past_key_7_in" NHWC --source_model_input_layout "past_value_7_in" NHWC
                 --source_model_input_layout "past_key_8_in" NHWC --source_model_input_layout "past_value_8_in" NHWC
                 --source_model_input_layout "past_key_9_in" NHWC --source_model_input_layout "past_value_9_in" NHWC
                 --source_model_input_layout "past_key_10_in" NHWC --source_model_input_layout "past_value_10_in" NHWC
                 --source_model_input_layout "past_key_11_in" NHWC --source_model_input_layout "past_value_11_in" NHWC
                 --source_model_input_layout "past_key_12_in" NHWC --source_model_input_layout "past_value_12_in" NHWC
                 --source_model_input_layout "past_key_13_in" NHWC --source_model_input_layout "past_value_13_in" NHWC
                 --source_model_input_layout "past_key_14_in" NHWC --source_model_input_layout "past_value_14_in" NHWC
                 --source_model_input_layout "past_key_15_in" NHWC --source_model_input_layout "past_value_15_in" NHWC
                 --source_model_input_layout "past_key_16_in" NHWC --source_model_input_layout "past_value_16_in" NHWC
                 --source_model_input_layout "past_key_17_in" NHWC --source_model_input_layout "past_value_17_in" NHWC
                 --source_model_input_layout "past_key_18_in" NHWC --source_model_input_layout "past_value_18_in" NHWC
                 --source_model_input_layout "past_key_19_in" NHWC --source_model_input_layout "past_value_19_in" NHWC
                 --source_model_input_layout "past_key_20_in" NHWC --source_model_input_layout "past_value_20_in" NHWC
                 --source_model_input_layout "past_key_21_in" NHWC --source_model_input_layout "past_value_21_in" NHWC
                 --source_model_input_layout "past_key_22_in" NHWC --source_model_input_layout "past_value_22_in" NHWC
                 --source_model_input_layout "past_key_23_in" NHWC --source_model_input_layout "past_value_23_in" NHWC
                 --source_model_input_layout "past_key_24_in" NHWC --source_model_input_layout "past_value_24_in" NHWC
                 --source_model_input_layout "past_key_25_in" NHWC --source_model_input_layout "past_value_25_in" NHWC
                 --source_model_input_layout "past_key_26_in" NHWC --source_model_input_layout "past_value_26_in" NHWC
                 --source_model_input_layout "past_key_27_in" NHWC --source_model_input_layout "past_value_27_in" NHWC
                 --source_model_input_layout "past_key_28_in" NHWC --source_model_input_layout "past_value_28_in" NHWC
                 --source_model_input_layout "past_key_29_in" NHWC --source_model_input_layout "past_value_29_in" NHWC
                 --source_model_input_layout "past_key_30_in" NHWC --source_model_input_layout "past_value_30_in" NHWC
                 --source_model_input_layout "past_key_31_in" NHWC --source_model_input_layout "past_value_31_in" NHWC
```

2. **Quantization (using qairt-quantizer)**

```
qairt-quantizer  --input_dlc model_float.dlc --output_dlc model_fp16w4.dlc
                 --weights_bitwidth 4 --keep_weights_quantized --pack_4_bit_weights
                 --float_bitwidth 16 --float_fallback --export_stripped_dlc
```

Step2 – Context-bin generation (offline preparation)

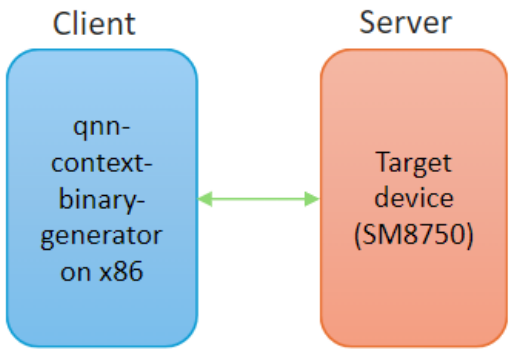
The context binary for LLaMA2-7b can be generated on x86(client) by connecting to device (server).

1. Launch server on target device

- Push server executable
  - adb push $QAIRT_SDK/bin/aarch64-android/qnn-gpu-target-server /data/local/tmp/
- Setup adb port forwarding, default port is '22598':
  - adb forward tcp:22598 tcp:22598
- Launch server
  - adb shell "/data/local/tmp/qnn-gpu-target-server"

The client (qnn-context-binary-generator) listens on the port '22598' and connects to the server and get required information from device and generate the context binary

Note: If target device is accessed remotely and not connected to the host, set the environment so the server is discoverable.
export TARGET_SERVER_SOCKET=<remote host>:<port>



2. Launch client on x86

```
$QAIRT_SDK/bin/x86_64-linux-clang/qnn-context-binary-generator
                --dlc_path <path to dlc>
                --backend libQnnGpu.so
                --model libQnnModelDlc.so
                --binary_file <name of serialized bin>
                --output_dir <output directory>
                --config_file <path to gpu_backend_ext.json>
```

gpu_backend_ext.json

```
{
  "backend_extensions": {
    "shared_library_path": "libQnnGpuNetRunExtensions.so",
    "config_file_path": "<path to gpu_options.json>"
  }
}
```

gpu_options.json

```
{
  "graph_names": [
    "<graph name>"
  ],
  "tuning_mode": true,
  "performance_cache_dir": "<path to cache database>",
  "precision_mode": "fp16",
  "perf_hint": "high"
}
```

Step3 – Execution using genie-t2t-run

Refer Genie tutorials/examples in QAIRT SDK documentation

- Push the required artifacts from QAIRT SDK to device
  - genie-t2t-run
  - libGenie.so
  - libQnnGpu.so
  - libQnnSystem.so
  - tokenizer.json
  - context binary generated from previous step

Execution

- adb shell
- cd <path on device>
- export LD_LIBRARY_PATH=<path to libraries>
- export PATH=<path to genie-t2t-run>
- genie-t2t-run -c <path to genie gpu config > -p "Tell me about Qualcomm"

Genie gpu config

```
{
  "dialog" : {
    "version" : 1,
    "type" : "basic",
    "context" : {
      "version" : 1,
      "size": 1024,
      "n-vocab": 32000,
      "bos-token": 1,
      "eos-token": 2
    },
    "sampler" : {
      "version" : 1,
      "seed" : 100,
      "temp" : 1.1,
      "top-k" : 40,
      "top-p" : 0.95,
      "greedy" : false
    },
    "tokenizer" : {
      "version" : 1,
      "path" : "<path to tokenizer.json>"
    },
    "engine" : {
      "version" : 1,
      "n-threads" : 3,
      "backend" : {
        "version" : 1,
        "type" : "QnnGpu"
      },
      "model" : {
        "version" : 1,
        "type" : "binary",
        "binary" : {
          "version" : 1,
          "ctx-bins" : [
            "<path to context binary>"
          ]
        }
      }
    }
  }
}
```

Qualcomm

Thank you!

Follow us on:

For more information, visit us at:

www.qualcomm.com & www.qualcomm.com/blog