

Appendix — Understanding Float-to-Int8 Conversion and Scaling**

Why Do We Convert float32 IQ Data to int8?

Most SDR systems process IQ data in high-precision floating-point (like `float32`), but this format is bulky (4 bytes per sample) and inefficient for storage/transmission. To reduce data size and meet standard SDR format expectations (like 8-bit IQ files), we **scale and convert** these float values into `int8`.

What is int8 and Why Use 127?

- `int8` is an 8-bit signed integer type
- It can hold values from **-128 to +127**
- We use **127** as the scale factor so that normalized float values from **-1.0 to +1.0** are mapped to the full `int8` range:

```
scaled = (float_values * 127).astype(np.int8)
```



This way:

- `-1.0` → `-127`
- `0.0` → `0`
- `1.0` → `+127`

This **maximizes resolution** while avoiding overflow (note: `+128` doesn't exist in `int8`).

The Real Meaning of "Detail"

Detail refers to how many unique levels (steps) you can preserve from the original signal. Here's why scaling is critical:

Scaling Factor	Usable int8 Levels	Signal Quality
10	-10 to +10 (21)	 Coarse
127	-127 to +127 (255)	 Fine

If you scale by 10, you're throwing away 90% of the potential precision of int8.

What is Quantization Error?

After scaling, float values must be **rounded** to the nearest integer. This introduces a small error:

```
quant_error = original_value - (scaled_value / scale_factor)
```

Smaller scale → bigger step size → higher error

Full scale (127) → smallest error possible for int8

Example

```
x = np.array([-1.0, -0.5, 0.0, 0.5, 1.0], dtype=np.float32)
print((x * 127).astype(np.int8))
# Output: [-127, -63, 0, 63, 127]
```

This keeps the shape of the waveform.

Compare that with:

```
print((x * 10).astype(np.int8))
# Output: [-10, -5, 0, 5, 10] – coarse!
```

What if float > 1.0?

This can cause overflow:

```
x = np.array([3.2])
print((x * 127).astype(np.int8))
# Output: wraps around – wrong result
```



Solutions:

1. **Normalize first:**

```
x = x / np.max(np.abs(x))
```

2. **Or clip values:**

```
x = np.clip(x, -1.0, 1.0)
```



Summary Table

Concept	What It Means
float32	Precise representation of IQ samples in software (32-bit float)
int8	Compact form for storage/transmission (8-bit signed integer)
Scaling	Expands float to use full int8 range
Quantization error	Loss of precision from rounding float → int
Clipping	Prevents overflow by limiting float values to −1.0 to +1.0



Closing Insight:

Even though your float signal is already sampled and quantized, converting to int8 is a **second-stage quantization** that trades off space for precision. To preserve signal shape, amplitude detail, and minimize loss, scaling by 127 is the practical and standardized method in IQ data workflows.

This step is critical for SDR data formatting, compression, visualization, and downstream DSP operations.



This appendix serves as a standalone educational reference for float-to-int8 scaling decisions in IQ data handling.