Encryption and Singing performance:

**Signing vs Encryption Time:**  
Dilithium’s signature generation is computationally heavier than Kyber’s key encapsulation (encryption) because Dilithium relies on structured lattice sampling and complex polynomial operations that dominate its runtime.

**Verification vs Decryption Time:**

Dilithium’s signature verification is significantly faster and less computationally intensive than its signing operation, on the other hand, Kyber decryption (decapsulation) involves polynomial arithmetic and number-theoretic transforms that are more computationally demanding than verification in Dilithium, explaining why verification is faster than decryption in your measurements.

**Signing is computationally heavier than verification**

Signing involves complex structured lattice sampling, rejection sampling, and polynomial arithmetic, which are computationally intensive. Verification mainly performs simpler linear algebraic checks and polynomial evaluations without the costly sampling steps, making it significantly faster.

So it appears that hybrid signing and verifying with ECDSA+ Dilithum2 is more computationally heavy than encryption and decrypting with ECDH + Kybers.

Explanation of Kyber Performance on Jetson Xavier NX which kyber768 appears to be faster than kyber512 and kyber1024

1. **Vectorization and Parallelism**  
   Kyber1024 uses larger polynomial dimensions (k=4, n=256) compared to Kyber512 (k=2, n=128) and Kyber768 (k=3, n=192). This larger structure allows better utilization of SIMD/vector instructions available on ARMv8.2 CPUs like the Carmel cores, which support NEON vector extensions. The wider data paths and increased parallelism can partially offset the higher computational complexity of Kyber1024, improving signing speeds relative to Kyber512 despite its larger parameter size.
2. **Memory Access and Cache Utilization**  
   The Jetson Xavier NX’s 8GB RAM and relatively large CPU caches facilitate efficient memory access. Kyber1024’s larger polynomial and key sizes lead to more predictable memory prefetching and fewer cache misses during modular arithmetic and polynomial multiplications. This reduces pipeline stalls and improves throughput, especially for signing operations that involve repeated modular reductions.
3. **CPU Frequency and Power Mode**  
   Operating in 20W 6-core mode balances performance and power consumption but may limit peak CPU frequency compared to higher power modes. Kyber1024’s heavier computations benefit from sustained CPU performance and vectorization, which your platform supports well, explaining why Kyber1024’s client signing speed remains competitive.

Sending with TLS 1.3 (python API):

As expected sending/receiving using only TLS 1.3 is faster than when using the additional hybrid security with hybrid encryption and hybrid signatures.

1. **TLS sendall() is a blocking call that returns only after the data is copied into the OS socket buffers and TLS record layer has processed it.** The TLS layer encrypts and authenticates the data before sending, so the CPU cost of TLS symmetric encryption and record framing affects sendall() timing. However, since your hybrid Kyber encryption and signing occur *before* calling sendall(), their CPU cost is *not* included in this timing.
2. **Higher Kyber levels increase CPU load on the client system overall.** Even though the hybrid encryption/signing is done before sendall(), the client CPU may be more heavily loaded with Kyber1024 than Kyber512. This can cause CPU contention or slower TLS stack processing, indirectly increasing the time sendall() takes to complete.
3. **TLS record layer overhead and internal buffering may be affected by system load and packet sizes.** While the encrypted payload size per chunk is constant, TLS adds headers and may fragment records differently depending on timing and system conditions, causing subtle delays in sendall().
4. **Python’s TLS API and socket buffers may introduce variability.** The Python ssl module wraps OpenSSL and the OS socket; if the socket send buffer is full or network congestion occurs, sendall() will block longer. Higher CPU usage from Kyber operations can reduce the rate at which the client processes outgoing data, causing buffer backpressure.
5. **Network stack scheduling and context switching overhead.** Increased CPU usage from Kyber hybrid crypto can cause more context switches or scheduling delays, slowing down TLS stack operations and increasing sendall() latency.