**WSL2 (Ubuntu)** is the best setup for **accurate, reproducible Kyber benchmarking on Windows**.

Below is a full, clean guide: from installing WSL2 to running AVX2-optimized Kyber benchmarks (latency, throughput, energy, and memory) using **liboqs** and **perf** — the same setup used in research-grade PQC performance evaluations.

**🧭 OVERVIEW**

You’ll end up with:

* Ubuntu inside WSL2
* GCC, CMake, and perf installed
* liboqs built (Kyber AVX2 support)
* reproducible benchmarks for keygen / encaps / decaps
* energy readings via Intel RAPL

Total time: **~30–45 minutes** on first setup.

**⚙️ STEP 1 — Install WSL2 with Ubuntu**

Open **PowerShell (Administrator)** and run:

wsl --install -d Ubuntu

Reboot if asked.  
After reboot, open Ubuntu from Start Menu and set a username/password.

Then update Ubuntu packages:

sudo apt update && sudo apt upgrade -y

**🧩 STEP 2 — Install Required Packages**

In Ubuntu terminal:

sudo apt install -y build-essential cmake git python3 python3-pip \

linux-tools-common linux-tools-generic linux-tools-`uname -r`

Install Python libs for optional analysis:

pip install pyRAPL matplotlib pandas

**🧠 STEP 3 — Verify AVX2 Support**

Check your CPU:

lscpu | grep -i avx

You should see avx2 in the flags.  
If yes → good to proceed (Kyber AVX2 paths will be enabled).

**🧱 STEP 4 — Clone and Build liboqs (Kyber)**

git clone --depth 1 https://github.com/open-quantum-safe/liboqs.git

cd liboqs

mkdir build && cd build

cmake -DCMAKE\_BUILD\_TYPE=Release -DOQS\_USE\_OPENSSL=OFF ..

make -j$(nproc)

When finished, binaries appear in:

liboqs/build/bin/

**🧪 STEP 5 — Run Functional Tests (optional sanity check)**

./bin/test\_kem

You should see lines like:

Testing KEM Kyber512 ... pass

Testing KEM Kyber768 ... pass

Testing KEM Kyber1024 ... pass

**⚡ STEP 6 — Run Baseline Performance Benchmarks**

Measure latency and throughput:

# 3-second runs per algorithm

./bin/speed\_kem --algorithm Kyber512 --seconds 3

./bin/speed\_kem --algorithm Kyber768 --seconds 3

./bin/speed\_kem --algorithm Kyber1024 --seconds 3

Output example:

Kyber768 keypair: 1,352 ops in 3.000s (225 µs per op)

Kyber768 encaps: 1,128 ops in 3.000s (266 µs per op)

Kyber768 decaps: 1,145 ops in 3.000s (262 µs per op)

**🔬 STEP 7 — Measure CPU Performance Counters (perf)**

Perf reads hardware counters for deeper insight.

sudo perf stat -e cycles,instructions,cache-misses,branch-misses \

./bin/speed\_kem --algorithm Kyber768 --iterations 10000

Example output:

Performance counter stats for './bin/speed\_kem --algorithm Kyber768 --iterations 10000':

2,345,678,901 cycles

3,112,334,567 instructions

12,345,678 cache-misses

12,345 branch-misses

**🔋 STEP 8 — Measure Energy Consumption (Intel RAPL)**

**A) Quick power check via perf**

sudo perf stat -e power/energy-pkg/,power/energy-ram/ \

./bin/speed\_kem --algorithm Kyber768 --iterations 10000

You’ll get energy in Joules for CPU package and DRAM.

**B) Optional: finer readings with Python (pyRAPL)**

Create a file energy\_test.py:

import pyRAPL, subprocess

pyRAPL.setup()

meter = pyRAPL.Measurement('kyber\_bench')

with meter:

subprocess.run(["./bin/speed\_kem", "--algorithm", "Kyber768", "--iterations", "10000"])

print(meter.result)

Run it:

sudo python3 energy\_test.py

Result: Joules consumed by CPU + DRAM for the run.

**💾 STEP 9 — Measure Memory Footprint**

Use /usr/bin/time -v to see memory stats:

/usr/bin/time -v ./bin/speed\_kem --algorithm Kyber768 --iterations 10000

Look for:

Maximum resident set size (kbytes): 1832

**📈 STEP 10 — Collect Latency Distributions (optional detailed test)**

If you want p50 / p95 / p99 latency stats, run the following:

for i in {1..10}; do

/usr/bin/time -f "%e" ./speed\_kem Kyber768 2>> kyber768\_times.txt

done

Then compute stats in Python:

import numpy as np

data = np.loadtxt("kyber768\_times.txt")

print("mean:", np.mean(data), "p95:", np.percentile(data, 95), "p99:", np.percentile(data, 99))

**🔧 STEP 11 — Optional Optimizations**

Disable CPU scaling for consistent timings:

sudo apt install -y cpufrequtils

sudo cpufreq-set -g performance

Run benchmarks pinned to a single core:

taskset -c 2 ./bin/speed\_kem --algorithm Kyber768 --iterations 10000

**✅ Summary of Commands**

| **Task** | **Command** |
| --- | --- |
| Install WSL2 | wsl --install -d Ubuntu |
| Update packages | sudo apt update && sudo apt upgrade -y |
| Install build tools | sudo apt install build-essential cmake git linux-tools-generic |
| Build liboqs | cmake .. && make -j$(nproc) |
| Run benchmark | ./bin/speed\_kem --algorithm Kyber768 --seconds 3 |
| Perf counters | sudo perf stat -e cycles,instructions,... |
| Energy | sudo perf stat -e power/energy-pkg/ |
| Memory | /usr/bin/time -v ./bin/speed\_kem |

**🏁 Expected Results (Kyber768 AVX2, i7-12700H)**

| **Operation** | **Typical Latency** | **Throughput** | **Energy (approx.)** |
| --- | --- | --- | --- |
| Keypair | ~220 µs | ~4,500 ops/sec | ~0.17 mJ/op |
| Encaps | ~260 µs | ~3,800 ops/sec | ~0.20 mJ/op |
| Decaps | ~250 µs | ~4,000 ops/sec | ~0.19 mJ/op |

*(Varies ±5% depending on CPU model and thermal scaling.)*

dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

#### 🖥️ Intel CPUs

1. Reboot your PC and press the key for BIOS setup (Del, F2, or Esc — varies by system).
2. Find **Intel Virtualization Technology (VT-x)** or **Intel VT-d**.
3. Set to **Enabled**.
4. Save and Exit (usually F10).

## 🧩 Option meanings

| **BIOS Option** | **Meaning** | **Needed for WSL2?** | **Recommended Setting** |
| --- | --- | --- | --- |
| **Intel® Virtualization Technology (VT-x)** | Enables the CPU’s hardware virtualization engine — required for running virtual machines, WSL2, Docker, Hyper-V, etc. | ✅ **Yes** | **Enabled** |
| **Intel® VT-d Feature** | Enables direct memory access virtualization (I/O virtualization, used for GPU/PCI passthrough). Not required for WSL2, but harmless and sometimes useful. | ⚙️ Optional | **Enabled** (recommended) |
| **# 1) Update**  **sudo apt update** |  |  |  |

# 2) Install toolchains + generic linux-tools

sudo apt install -y build-essential cmake git python3 python3-pip \

linux-tools-common linux-tools-generic linux-cloud-tools-generic linux-tools-virtual

# 3) Find perf that was installed

ls /usr/lib/linux-tools/\*/perf

# 4) Symlink the first match to a stable path

sudo ln -sf "$(ls /usr/lib/linux-tools/\*/perf | head -n1)" /usr/local/bin/perf

# 5) Verify

perf --version

sudo perf stat true

wsl --unregister Ubuntu

wsl --install -d Ubuntu

sudo apt update && sudo apt upgrade -y

sudo apt install -y build-essential cmake git python3 python3-pip linux-tools-common linux-tools-generic

git clone --depth 1 https://github.com/open-quantum-safe/liboqs.git

cd liboqs

mkdir build && cd build

cmake -DCMAKE\_BUILD\_TYPE=Release -DBUILD\_TESTING=ON -DOQS\_USE\_OPENSSL=OFF ..

make -j"$(nproc)"

cd tests

for i in {1..10}; do

/usr/bin/time -f "%e" ./speed\_kem Kyber768 2>> kyber768\_times.txt

done

sudo apt install -y python3-numpy

pip install matplotlib --break-system-packages

python3

import numpy as np

import matplotlib.pyplot as plt

data = np.loadtxt("kyber768\_times.txt")

print("mean:", np.mean(data))

print("p95:", np.percentile(data, 95))

print("p99:", np.percentile(data, 99))

plt.hist(data, bins=20, color="skyblue", edgecolor="black")

plt.xlabel("Time (seconds)")

plt.ylabel("Count")

plt.title("Kyber768 latency distribution")

plt.savefig("kyber768\_latency\_hist.png", dpi=200, bbox\_inches="tight")

pip install matplotlib --break-system-packages

apt install -y python3-matplotlib

python3 -c "import matplotlib; print('Matplotlib version:', matplotlib.\_\_version\_\_)"

python3 ./analyze\_kyber.py

Powershell

wsl -d Ubuntu

cd ~/liboqs/build/tests