MLX-RS Performance Analysis: GLM-4.5 MoE

Executive Summary

Current Status: ✅ **Performance parity achieved!**

Metric	Before	After
Rust vs Python	**3.24x slower**	**~1.0x (parity)**
127 tokens	958ms	598ms
Gap	60% slower below 128 tokens	Eliminated

Performance Comparison (After MLX v0.30.1 Update)

Seq Len	Python (ms)	Rust (ms)	Difference
32	267.4	263.3	**Rust 1.5% faster**
64	405.2	392.1	**Rust 3.2% faster**
127	615.5	598.4	**Rust 2.8% faster**
128	616.7	601.7	**Rust 2.4% faster**
256	1014.5	1022.1	Python 0.7% faster
512	1854.4	1737.8	**Rust 6.3% faster**

Conclusion: Rust is now at performance parity with Python, and often slightly faster.

Root Causes Identified and Fixed

1. ✅ 128-Token Threshold Bug (FIXED)

Problem: Rust was 60% slower for sequences < 128 tokens.

Root Cause: MLX v0.29.1 had a bug in `scaled_dot_product_attention` with causal masking that caused suboptimal memory copies for sequences < 128 tokens. Fixed in MLX v0.30.0 (PR

#2563: "fix copies in sdpa").

Solution: Updated mlx-c submodule from v0.3.0 (MLX v0.29.1) to v0.4.1 (MLX v0.30.1).

2. ✅ API Changes for MLX v0.30.1 (FIXED)

Required changes to mlx-rs bindings:

Component	Change
SDPA	Mask changed from `VectorArray` to single `Array`
quantize	`group_size` / `bits` now `mlx_optional_int_`
quantized_matmul	`group_size` / `bits` now `mlx_optional_int_`
dequantize	Added optional `dtype` parameter
gather_qmm	`group_size` / `bits` now `mlx_optional_int_`

3. ✅ macOS Tahoe Beta Compatibility (FIXED)

Problem: MLX v0.30.1 uses Metal 4.0 and `__builtin_available(macOS 26)` which caused build/link errors on Xcode beta.

Solution: Added patches in `build.rs`:

- Force Metal 3.2 in `device.cpp` (Metal 4.0 not supported in current Xcode beta)
- Disable NAX feature in `device.h` (avoids `___isPlatformVersionAtLeast` link error)

Files Modified

mlx-sys (bindings)

1. **`src/mlx-c`** - Updated submodule to v0.4.1
2. **`build.rs`** - Added:
   - Custom CMake build process with patching support
   - `patch_metal_version()` function to patch device.cpp and device.h
   - macOS 15.0 deployment target for consistency

mlx-rs (Rust API)

1. **`src/fast.rs`**

- Added `Guarded` trait import
- Updated SDPA to use single `Array` mask instead of `VectorArray`
- Changed `as_mode_and_masks()` to `as_mode_and_mask_ptr()`

2. **`src/ops/quantization.rs`**
   - Added `optional_int()` helper function
   - Added `optional_dtype_none()` helper function
   - Updated all quantization functions to use new optional types

Previous Analysis (Historical Context)

The following analysis was done before identifying the MLX version issue:

Individual Operations are NOT the Problem

Operation	Rust	Python	Ratio
SwiGLU	0.54ms	0.56ms	0.96x (Rust faster!)
RMSNorm	0.32ms	0.31ms	1.05x
SDPA	0.24ms	0.32ms	0.73x (Rust faster!)
Matmul	0.47ms	0.41ms	1.14x
MoE Routing	0.26ms	0.25ms	1.03x

Conclusion: Individual MLX operations perform nearly identically in Rust and Python.

Graph Building is NOT the Problem

Metric	Rust	Python
Graph building time	1.54ms	1.61ms

Conclusion: Graph construction is fast in both implementations (~1.5ms).

Benchmark Commands

```
# Single sequence length test (Rust)
cargo run --release --example single_seqlen -- 127

# Single sequence length test (Python)
python python_single_seqlen.py 127

# Full sweep
for len in 32 64 127 128 256 512; do
  cargo run --release --example single_seqlen -- $len
  python python_single_seqlen.py $len
done
```

Conclusion

The 3.24x performance gap has been **completely eliminated**. The root cause was using an older version of MLX (v0.29.1) that had a bug in SDPA with causal masking. After updating to MLX v0.30.1 and fixing the required API changes, Rust now achieves **performance parity with Python**, and is often slightly faster.

Key Takeaways

1. **Keep MLX bindings up to date** - Performance bugs in upstream MLX can cause significant regressions
2. **The 128-token threshold was a real bug** - Not a quirk of the Rust implementation
3. **FFI overhead is negligible** - The C bindings add no measurable overhead
4. **Rust can match Python performance** - With proper bindings, there's no inherent disadvantage

Next Steps (Optional Improvements)

1. **Remove unused** `VectorArray::from_ptr` - Dead code warning
2. **Clean up build.rs** - Remove unused `cmake::Config` import
3. **Consider contributing patches upstream** - Metal 4.0 compatibility for Xcode beta