ARCHITECTURE DEEP DIVE

Inference Optimization

Triton kernels, speculative decoding, and memory optimization techniques. How to achieve 274x speedups without sacrificing output quality.

The Key Insight

Most optimization techniques sacrifice quality for speed. Speculative decoding is different: it's mathematically equivalent to standard decoding. Same outputs, dramatically faster. No compromises.

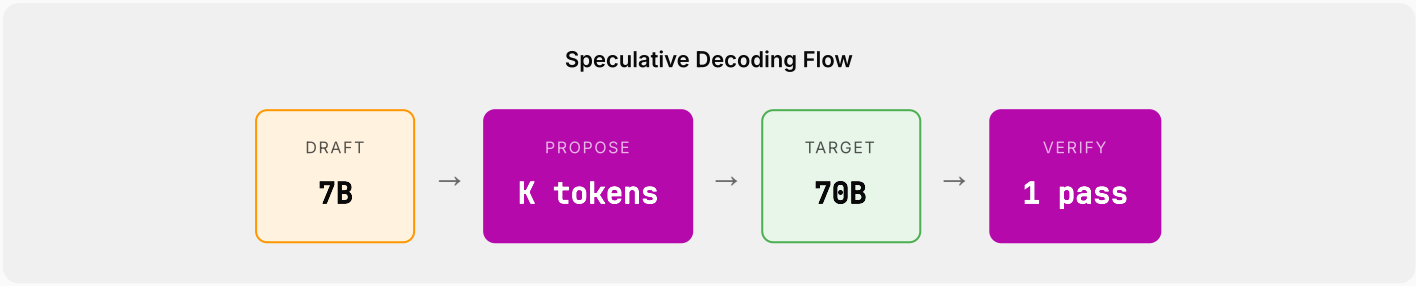
274x	3-5x	60%	0%
Max speedup	Typical chat speedup	Cost reduction	Quality loss

IN THIS WHITEPAPER

01 - SPECULATIVE DECODING

Draft-then-verify for dramatic speedups

Speculative decoding uses a small, fast draft model to propose multiple tokens, then verifies them with the target model in a single forward pass. Same outputs, fewer forward passes.



Mathematical Equivalence

Speculative decoding produces the exact same output distribution as standard autoregressive decoding. The verification step uses rejection sampling to ensure correctness. No quality loss, guaranteed.

Acceptance Rate

When draft tokens match target distribution, they're accepted. Typical acceptance rates are 60-80%. Higher rates = more speedup. Rates depend on draft model quality and task domain.

Speculation Length (K)

Number of tokens to propose per iteration. Higher K = more potential speedup but lower acceptance rate. Optimal K varies by workload. Typically 4-8 for chat, 16-32 for code.

Draft Model Selection

Smaller model from same family works best. Llama 7B drafts for Llama 70B. Mistral 7B drafts for Mixtral. Fine-tuned drafts can achieve higher acceptance rates.

Why 274x for Verified Synthesis?

Code generation with verification achieves extreme speedups because: (1) draft model proposes code, (2) verifier checks correctness instantly via execution, (3) if wrong, restart is cheap. The target model only runs for accepted sequences.

Key insight: Verification doesn't need LLM reasoning. Execution is deterministic and fast.

02 - CUSTOM TRITON KERNELS

Hand-optimized GPU operations

EXAMPLE: FUSED ATTENTION KERNEL (TRITON)

```
# Fused attention eliminates memory round-trips
@triton.jit
def fused_attention(q, k, v, output, ...):
    # Compute QK^T, softmax, and V multiplication
    # in a single kernel without intermediate writes
    acc = tl.zeros([BLOCK_M, BLOCK_N], dtype=tl.float32)
    # ... optimized implementation
```

03 - MEMORY OPTIMIZATION

Reducing the memory bottleneck

LLM inference is often memory-bound, not compute-bound. Optimizing memory access patterns is critical for throughput.

KV Cache Optimization

Key-value caching stores previous attention computations. Paged attention (vLLM) enables efficient memory management. Dynamic allocation prevents OOM on long sequences.

Kernel Fusion

Combine multiple operations into single GPU kernels. Eliminates memory round-trips between operations. Custom fused kernels for attention, MLP, and layer norm.

Continuous Batching

Dynamic request scheduling maximizes GPU utilization. Requests join/leave batches as they complete. No wasted compute waiting for long sequences.

04 - QUANTIZATION TECHNIQUES

Precision tradeoffs

TECHNIQUE	BITS	MEMORY REDUCTION	QUALITY IMPACT
FP16/BF16	16	50%	None
INT8	8	75%	Minimal (<0.5%)
INT4 (GPTQ/AWQ)	4	87.5%	Small (1-3%)
FP8	8	75%	None (with H100)

Post-Training Quantization

Apply quantization after training. GPTQ and AWQ are popular methods. Requires calibration dataset. Usually 4-bit with minimal quality loss.

Mixed Precision

Keep sensitive layers at higher precision. Attention and final layers often need FP16. MLP layers can often use INT8/INT4. Profile to find optimal mix.

H100 FP8: Best of Both Worlds

NVIDIA H100's native FP8 support provides 75% memory reduction with no quality loss. 2x throughput vs FP16. If you have H100s, use FP8.

05 - BENCHMARKS

Real-world performance

Benchmarks on Llama 2 70B with A100 80GB. Your results will vary based on hardware and workload.

CHAT WORKLOAD

3.2x

Latency reduction with speculative decoding

CODE GENERATION

5.8x

Throughput increase with custom kernels

BATCH PROCESSING

2.4x

Cost reduction with continuous batching

OPTIMIZATION	LATENCY IMPACT	THROUGHPUT IMPACT	MEMORY IMPACT
Speculative Decoding (K=8)	**-68%**	+45%	+15% (draft model)
Fused Attention Kernels	**-22%**	+28%	No change
Continuous Batching	Variable	**+85%**	No change
INT8 Quantization	-5%	+40%	**-50%**

06 - IMPLEMENTATION WITH ACCELERATE

Getting started

Start with Profiling

We begin with a comprehensive audit of your inference pipeline. Identify bottlenecks, measure baseline performance, and quantify optimization opportunities.

Workload-Specific Tuning

Every workload is different. Chat needs low latency. Batch processing needs throughput. We tune speculation length, batch size, and kernel parameters for your use case.

Quality Validation

Rigorous testing ensures no quality regression. We run your eval suite before and after. Statistical validation proves equivalence. No surprises in production.

Minimal Code Changes

Drop-in integration with your existing pipeline. Python SDK wraps your inference calls. Usually just a few lines of code to integrate and see immediate improvements.

Ready to optimize?

Get a performance audit for your inference workload.

Request Audit | Learn More