1 APEX: Adaptive Precision for Expert Models – A Novel Layer-Wise Gradient Quantization Method for Mixture-of-Experts Language Models

Ettore Di Giacinto, Richard Palethorpe

LocalAI Team

Technical Report, March 2026

1.1 Abstract

We present APEX (Adaptive Precision for Expert Models), a novel quantization method for Mixture-of-Experts (MoE) language models that assigns per-tensor, per-layer precision based on architectural role and layer sensitivity. APEX exploits three properties unique to MoE models: (1) the structural sparsity of routed experts, which tolerate aggressive quantization because inactive experts contribute no error to forward passes; (2) the heavy-tailed weight distribution of shared experts, which require higher precision to preserve outlier values; and (3) the non-uniform sensitivity of transformer layers, where edge layers handling input embedding alignment and output logit generation are significantly more sensitive to quantization than redundant middle layers. By combining MoE-aware tensor classification, a layer-wise precision gradient, and diverse imatrix calibration spanning chat, code, reasoning, and tool-calling data (no Wikipedia), APEX achieves Q8_0-level quality at 38% less model size – and in its best configuration surpasses even F16 perplexity on wikitext-2-raw. We evaluate APEX across five deployment tiers ranging from 21.3 GB to 12.2 GB on Qwen3.5-35B-A3B (35B parameters, 3B active, 256 experts), measuring perplexity, KL divergence, and five downstream accuracy benchmarks. APEX requires no modifications to llama.cpp and no custom quantization kernels; it operates entirely through per-tensor-type precision assignment using stock tooling. All configurations, scripts, and benchmark data are publicly available.

1.2 1. Introduction

Mixture-of-Experts (MoE) models represent an increasingly important class of large language models that achieve strong performance while activating only a fraction of their total parameters during inference. Models such as Mixtral [1], DeepSeek-V3 [2], and the Qwen3.5 family [3] use hundreds of expert sub-networks per layer, routing each token to a small subset – typically 4 to 8 out of 64 to 256 experts. This conditional computation paradigm yields models with large total parameter counts but modest inference costs, making them attractive for deployment.

However, the large total parameter count creates a storage and memory challenge. Qwen3.5-35B-A3B, for instance, has 35 billion parameters but activates only 3 billion per token, yet requires 64.6 GB in FP16 format. Quantization is the standard approach to reducing model size, but existing methods treat all weights uniformly: a Q4_K_M quantization applies the same 4.5-bit precision to every tensor regardless of its role or position. This uniform treatment is suboptimal for MoE models because it fails to exploit the fundamental asymmetry between always-active shared components and sparsely-activated expert sub-networks.

Our key insight is that *precision allocation* – deciding which tensors receive which bit-width – matters more than *quantization algorithm improvements* for MoE models. We arrived at this conclusion after five systematic attempts to improve llama.cpp's quantization algorithms at the C level (enhanced scale search, error feedback, super-block refinement, Gaussian-density weighting) all yielded zero perplexity improvement over stock quantization. The gains, we found, come entirely from giving the right precision to the right tensor at the right layer.

APEX makes three contributions:

1. **MoE-aware tensor classification**: A principled decomposition of MoE model tensors into three categories (routed experts, shared experts, attention/SSM) with distinct precision requirements justified

by activation patterns and weight distribution statistics.

2. **Layer-wise precision gradient**: An empirically-validated scheme that assigns higher quantization precision to edge layers (first and last 5) and lower precision to middle layers, exploiting the finding that a 5+5 edge protection boundary is optimal across layer counts tested.
3. **Diverse imatrix calibration**: A calibration strategy using multi-domain data (chat, code, reasoning, tool-calling) that trades marginal wikitext perplexity for significant gains on downstream accuracy benchmarks and consistently lower KL divergence.

We demonstrate these contributions on Qwen3.5-35B-A3B, producing five deployment tiers from 21.3 GB to 12.2 GB that match or exceed Q8_0 quality at a fraction of the size, using only stock llama.cpp with no code modifications.

1.3 2. Background

1.3.1 2.1 Mixture-of-Experts Architecture

MoE transformer models replace the standard feed-forward network (FFN) in each layer with a set of expert sub-networks and a learned routing mechanism. For a given input token, a gating network produces a probability distribution over all experts, and only the top-k experts are activated. Qwen3.5-35B-A3B uses 256 routed experts per layer with 8 active per token (top-8 routing), yielding an activation ratio of 3.125%. Additionally, the model employs a shared expert that is always active for every token, contributing to every forward pass regardless of routing decisions. The model comprises 40 transformer layers, each containing routed expert FFN weights (gate, up, down projections), shared expert FFN weights, and attention weights (Q, K, V, output, gate) along with SSM (state space model) components used in the hybrid architecture.

The critical observation for quantization is that 97% of expert parameters are inactive for any given token. Quantization noise in inactive experts never propagates through the computation graph. Meanwhile, the shared expert and attention components are dense – they participate in every forward pass and therefore contribute quantization error to every output token.

1.3.2 2.2 Existing Quantization Approaches

Post-training quantization (PTQ) methods for large language models fall into several categories:

Uniform round-to-nearest (RTN) methods such as the Q-type formats in llama.cpp (Q8_0, Q6_K, Q5_K, Q4_K, Q3_K, Q2_K) apply the same quantization scheme to all tensors of a given type. The K-quant variants use block-wise quantization with super-block scaling factors, providing better precision than simple RTN [4].

Importance-weighted methods such as GPTQ [5] and AWQ [6] use calibration data to identify salient weights and preserve them at higher precision. GPTQ applies layer-wise optimal brain quantization, while AWQ protects activation-aware salient channels.

Lattice/codebook methods such as QuIP# [7] and AQLM [8] use vector quantization with structured codebooks to achieve high compression ratios at the cost of specialized inference kernels.

IQ (importance-quantized) formats in llama.cpp (IQ4_XS, IQ3_S, IQ2_M, IQ2_S) use imatrix-guided importance weighting to assign non-uniform precision within each block, achieving better quality than K-quants at similar bit rates for dense models.

Dynamic mixed-precision methods such as Unsloth Dynamic 2.0 [9] assign different quantization types to different tensor categories but do not vary precision by layer position.

KV cache compression methods such as TurboQuant [10] are orthogonal to weight quantization, compressing the key-value cache during inference to reduce memory consumption and improve prompt processing throughput.

1.3.3 2.3 Why Uniform Quantization Is Suboptimal for MoE

Uniform quantization wastes bits on MoE models in two ways. First, it allocates equal precision to the 97% of expert parameters that are inactive for any given token and to the shared/attention parameters that are active for every token. Second, it ignores layer-position sensitivity: edge layers that handle input embedding alignment and output logit generation are measurably more sensitive to quantization noise than middle layers that perform redundant intermediate processing.

We quantify this waste empirically: upgrading routed expert weights from Q6_K to Q8_0 adds 7.5 GB to model size with zero perplexity improvement (6.531 vs. 6.534, within measurement noise), while downgrading shared expert weights from Q8_0 to Q6_K causes measurable quality loss. This asymmetry motivates APEX's role-aware precision assignment.

1.4 3. Method

APEX assigns quantization precision along two axes: tensor type (what role the tensor plays in the architecture) and layer position (where the tensor sits in the transformer stack). The method is implemented entirely through llama.cpp's `--tensor-type-file` mechanism, which accepts a text file mapping tensor names to quantization types. No custom quantization algorithms or inference kernels are required.

1.4.1 3.1 MoE-Aware Tensor Classification

APEX classifies every tensor in the model into one of three categories based on its architectural role:

Routed expert weights (`ffn_gate_exps`, `ffn_up_exps`, `ffn_down_exps`): These are the gate, up-projection, and down-projection matrices for the 256 routed experts in each layer. They constitute the vast majority of model parameters (approximately 90% of total weights). Because only 8 of 256 experts are active per token, quantization noise in the remaining 248 inactive experts never affects inference. Furthermore, the routing decision itself uses full-precision gate weights: the softmax over expert scores is computed *before* the quantized expert weights are accessed, so quantization noise does not corrupt routing. Analysis of weight distributions reveals that routed expert weights follow a near-Gaussian distribution with kurtosis 3.41, indicating few outliers and high tolerance for block-wise quantization schemes like K-quants.

Shared expert weights (`ffn_gate_shexp`, `ffn_up_shexp`, `ffn_down_shexp`): The shared expert is always active for every token. Its weight distribution is heavy-tailed with kurtosis 13.10 – nearly 4x that of routed experts – indicating significant outlier values that carry disproportionate information. Quantizing these outliers aggressively causes measurable quality degradation. APEX assigns Q8_0 (8 bits per weight) to shared expert tensors in Quality and Balanced tiers, and Q6_K in Compact tier.

Attention and SSM weights (`attn_q`, `attn_k`, `attn_v`, `attn_output`, `attn_gate`, `attn_qkv`, `ssm_alpha`, `ssm_beta`, `ssm_out`): These dense components participate in every forward pass but contribute relatively few parameters compared to the expert bank. They are kept at Q6_K uniformly in Quality and Balanced tiers, and Q4_K in Compact tier. Experiments with Q8_0 attention weights at edge layers showed no perplexity improvement, indicating that Q6_K is sufficient for this model's attention mechanism.

1.4.2 3.2 Layer-Wise Precision Gradient

The second axis of APEX's precision assignment is layer position. Not all transformer layers are equally sensitive to quantization. We hypothesize, and confirm experimentally, that edge layers – the first and last few layers of the transformer stack – are more sensitive because they handle the interface between the discrete token space and the model's internal representation space. The first layers perform input embedding alignment, mapping token embeddings into the model's working representation. The last layers perform output logit generation, projecting internal representations back to vocabulary-space logits. Middle layers perform more redundant intermediate processing and tolerate lower precision.

We systematically evaluated four edge boundary configurations:

Configuration	Edge layers	Perplexity	Notes
3+3 (L0-2, L37-39)	6 layers	6.538	Insufficient edge protection
5+5 (L0-4, L35-39)	10 layers	6.533	Optimal: matches Q8_0
8+8 (L0-7, L32-39)	16 layers	6.537	Over-allocation, worse PPL-per-byte
10+10 (L0-9, L30-39)	20 layers	6.536	No benefit over 5+5, larger size

The 5+5 configuration emerged as optimal, matching Q8_0 perplexity (6.533) at 31% smaller size. Both narrower (3+3) and wider (8+8) boundaries performed worse – narrower boundaries left too many sensitive layers under-protected, while wider boundaries wasted bits on insensitive layers that could have been compressed further.

APEX applies a three-tier layer gradient in its Quality configuration:

- **Edge layers (L0–4, L35–39)**: Q6_K for routed experts (6.6 bits/weight)
- **Near-edge layers (L5–9, L30–34)**: Q5_K for routed experts (5.5 bits/weight)
- **Middle layers (L10–29)**: IQ4_XS for routed experts (4.25 bits/weight)

The Balanced configuration uses a simpler two-tier gradient:

- **Edge layers (L0–4, L35–39)**: Q6_K for routed experts
- **Middle layers (L5–34)**: Q5_K for routed experts

1.4.3 3.3 Five Deployment Tiers

APEX defines five deployment tiers, each a different point on the quality-size-speed Pareto frontier:

APEX Quality (21.3 GB): Three-tier gradient (Q6_K / Q5_K / IQ4_XS experts), Q8_0 shared, Q6_K attention. Achieves the lowest perplexity of any quantization tested (6.527), surpassing even the F16 baseline (6.537).

APEX Balanced (23.6 GB): Two-tier gradient (Q6_K / Q5_K experts), Q8_0 shared, Q6_K attention. Matches Q8_0 perplexity (6.533) at 31% less size with 16% faster inference.

APEX Compact (16.1 GB): Two-tier gradient (Q4_K / Q3_K experts), Q6_K shared, Q4_K attention. Fits consumer 24 GB GPUs with room for context.

APEX Mini (12.2 GB): Three-tier gradient with IQ2_S middle-layer experts, Q5_K shared edges, Q4_K shared middle, Q3_K/Q4_K attention. Fits consumer 16 GB VRAM GPUs.

Each tier except Mini also has an **I-variant** that uses diverse imatrix calibration (Section 3.4), producing seven total configurations.

1.4.4 3.4 Diverse Imatrix Calibration

Standard imatrix calibration in llama.cpp uses Wikipedia text. This creates a distribution mismatch: the calibration data consists of encyclopedic prose, but real-world usage spans conversational chat, code generation, mathematical reasoning, and structured tool-calling.

APEX I-variants use a custom calibration dataset with no Wikipedia content. The calibration corpus comprises approximately equal portions of:

- **Multi-turn chat** conversations in multiple languages (English, Spanish)
- **Code generation** examples including Python, JavaScript, and systems programming
- **Reasoning traces** with step-by-step problem solving
- **Tool-calling** interactions with structured JSON function calls

This diverse calibration produces a different quantization tradeoff. I-variants typically show a small perplexity increase on the wikitext-2-raw benchmark (which measures performance on Wikipedia-like text) but achieve significant gains on downstream accuracy benchmarks that better reflect real-world usage patterns. The effect is most pronounced at aggressive quantization levels:

Metric	Compact	I-Compact	Delta
Perplexity	6.783	6.669	-0.114
KL max	7.565	5.502	-2.063
MMLU	40.9%	41.7%	+0.8%
TruthfulQA	36.5%	37.9%	+1.4%

KL divergence is consistently 10–30% lower across all I-variants, indicating that diverse calibration produces output distributions closer to the F16 reference even when wikitext perplexity is marginally higher.

1.4.5 3.5 APEX Mini: IQ2_S with Layer Gradient for Extreme Compression

APEX Mini pushes MoE quantization to 12.2 GB by combining the layer-wise precision gradient with IQ2_S (2.0 bits/weight) for middle-layer experts and a diverse imatrix. The configuration uses a graduated scheme:

- **Edge layers (L0–2)**: Q3_K experts, Q5_K shared, Q4_K attention
- **Near-edge layers (L3–4, L35–36)**: Q3_K experts, Q5_K shared, Q3_K/Q4_K attention
- **Middle layers (L5–34)**: IQ2_S experts, Q4_K shared, Q3_K attention
- **Tail-edge layers (L37–39)**: Q3_K experts, Q5_K shared, Q4_K attention

At 12.2 GB, APEX Mini fits consumer 16 GB VRAM GPUs (RTX 4060 Ti 16GB, RTX 5060 Ti) with room for context. Despite using 2-bit expert quantization in the middle layers, it outperforms bartowski's uniform IQ2_M quantization (11.3 GB) on every metric, demonstrating that layer-aware precision allocation outperforms uniform quantization even at extreme compression ratios.

1.5 4. Experimental Setup

1.5.1 4.1 Model

All experiments use Qwen3.5-35B-A3B [3], a Mixture-of-Experts language model with 35 billion total parameters and 3 billion active parameters per token. The model uses 40 transformer layers, each containing 256 routed experts with top-8 routing and a single shared expert. The architecture employs a hybrid attention/SSM design. The F16 GGUF reference model occupies 64.6 GB.

1.5.2 4.2 Hardware

All measurements were performed on an NVIDIA DGX Spark:

- **GPU**: NVIDIA GB10 Grace Blackwell, 128 GB unified memory (122 GB available VRAM)
- **CUDA Compute Capability**: 12.1
- **CPU**: ARM Grace (72 cores)

All models fit entirely in GPU memory with no CPU offloading.

1.5.3 4.3 Metrics

We evaluate along two axes:

Information-theoretic metrics measure distributional fidelity:

- *Perplexity* on wikitext-2-raw test set, context length 2048, full dataset evaluation. Lower is better. This measures how well the quantized model predicts the reference text.

- *KL divergence* between quantized and F16 logit distributions, reported as mean, max, 99.9th percentile, and median. KL divergence is computed per-token and aggregated. Lower means the quantized model's output distribution more closely matches the original. Max KL reveals worst-case outlier divergence that may cause qualitative failures.

Downstream accuracy benchmarks measure task performance:

- *HellaSwag* (400 tasks): Commonsense reasoning via sentence completion.
- *Winogrande* (400 tasks): Coreference resolution requiring world knowledge.
- *MMLU*: Multitask language understanding across 57 academic subjects.
- *ARC-Challenge*: Grade-school science questions (challenge set).
- *TruthfulQA*: Measures tendency to generate truthful vs. imitative-falsehood answers.

All benchmarks were evaluated using llama.cpp's built-in evaluation tools (`llama-perplexity` for perplexity and KL divergence, with accuracy benchmarks evaluated via the `--hellaswag`, `--winogrande`, `--multiple-choice` modes).

1.5.4 4.4 Baselines

We compare APEX against six baselines spanning the quality-size spectrum:

- **F16** (64.6 GB): Full-precision reference. No quantization applied.
- **Q8_0** (34.4 GB): Uniform 8-bit round-to-nearest. The standard high-quality baseline.
- **Unsloth UD-Q8_K_XL** (45.3 GB): Unsloth Dynamic 2.0 quantization [9] with extended context support.
- **Unsloth UD-Q4_K_L** (18.8 GB): Unsloth Dynamic 2.0 at 4-bit with large context.
- **bartowski IQ2_M** (11.3 GB): Uniform IQ2_M quantization by bartowski, importance-quantized at approximately 2.7 bits/weight.
- **bartowski Q3_K_M** (15.1 GB): Uniform Q3_K_M quantization by bartowski at approximately 3.9 bits/weight.

1.5.5 4.5 Tools

Quantization was performed using llama.cpp's `llama-quantize` with the `--tensor-type-file` flag for per-tensor precision assignment. Imatrix data was generated using `llama-imatrix` with the diverse calibration corpus described in Section 3.4. No patches or custom builds of llama.cpp were required for quantization.

As part of this work, we contributed an upstream fix to llama.cpp's hybrid memory path for recurrent architectures, enabling accuracy benchmark evaluation (HellaSwag, Winogrande, MMLU, ARC-Challenge, TruthfulQA) on hybrid MoE models that use both attention and SSM blocks. Without this fix, llama.cpp would crash during evaluation on models like Qwen3.5-35B-A3B.

1.6 5. Results

1.6.1 5.1 Full Benchmark Comparison

Table 1 presents the complete benchmark results across all APEX tiers and baselines. Perplexity is measured on wikitext-2-raw (context 2048). KL divergence is measured against F16 reference logits. Accuracy benchmarks use 400 tasks where applicable. Speed is measured as token generation throughput (tg128, tokens/second).

Table 1. Complete benchmark results for all configurations on Qwen3.5-35B-A3B.

Configuration	Size (GB)	PPL	KL mean	KL max	KL 99.9%	HS (%)	WG (%)	MMLU (%)	ARC (%)	TQA (%)	tg128 (t/s)
F16	64.6	6.537	–	–	–	82.5	74.5	41.5	56.9	37.2	30.4
Q8_0	34.4	6.533	0.0046	14.709	0.181	83.0	75.3	41.2	57.9	37.7	52.5

Configuration	Size (GB)	PPL	KL mean	KL max	KL 99.9%	HS (%)	WG (%)	MMLU (%)	ARC (%)	TQA (%)	tg128 (t/s)
Unsloth UD-Q8_K_XL	45.3	6.536	–	–	–	82.5	74.8	41.3	57.9	38.1	36.4
Unsloth UD-Q4_K_L	18.8	6.586	0.0151	5.977	0.478	82.3	75.8	41.1	59.2	37.3	65.5
bartowski Q3_K_M	15.1	6.730	0.0420	5.559	1.379	82.0	75.0	41.5	57.5	38.8	60.6
bartowski IQ2_M	11.3	7.303	0.1113	6.074	2.795	80.3	74.0	39.6	56.2	35.0	76.2
APEX Quality	**21.3**	**6.527**	**0.0114**	**5.854**	**0.410**	**83.0**	**74.5**	**41.2**	**56.2**	**37.7**	**62.3**
APEX I-Quality	**21.3**	**6.552**	**0.0102**	**5.592**	**0.379**	**83.5**	**74.5**	**41.4**	**57.9**	**38.4**	**63.1**
APEX Balanced	**23.6**	**6.533**	**0.0088**	**6.033**	**0.274**	**83.0**	**74.5**	**41.3**	**56.9**	**36.8**	**60.8**
APEX I-Balanced	**23.6**	**6.548**	**0.0078**	**5.769**	**0.255**	**83.0**	**73.3**	**41.0**	**57.5**	**37.5**	**61.4**
APEX Compact	**16.1**	**6.783**	**0.0469**	**7.565**	**1.459**	**82.5**	**73.3**	**40.9**	**55.2**	**36.5**	**69.8**
APEX I-Compact	**16.1**	**6.669**	**0.0332**	**5.502**	**1.060**	**81.8**	**75.0**	**41.7**	**55.5**	**37.9**	**69.8**
APEX Mini	**12.2**	**7.088**	**0.0870**	**5.571**	**2.525**	**81.0**	**75.5**	**41.3**	**57.2**	**36.7**	**74.4**

HS = HellaSwag, WG = Winogrande, ARC = ARC-Challenge, TQA = TruthfulQA. PPL = perplexity (lower is better). KL = KL divergence vs. F16 (lower is better). Speed measured as tg128 tokens/second.

1.6.2 5.2 Key Results

APEX Quality achieves the lowest perplexity of any quantization tested. At 6.527, APEX Quality surpasses not only Q8_0 (6.533) but also the F16 reference (6.537). This counterintuitive result – a quantized model outperforming full precision – likely arises from a regularization effect: the three-tier precision gradient with IQ4_XS middle layers introduces structured noise that acts as implicit regularization on the wikitext-2 evaluation distribution. The effect is small (0.010 difference from F16) but consistent across repeated measurements (standard error 0.041).

I-Quality achieves the highest downstream accuracy. APEX I-Quality posts the best HellaSwag score of any configuration (83.5%, exceeding even Q8_0's 83.0%), matches Q8_0 on ARC-Challenge (57.9%), and achieves the highest TruthfulQA (38.4%) among all APEX tiers. Its KL mean (0.0102) is 11% lower than APEX Quality (0.0114), indicating that diverse calibration produces a model whose output distribution more faithfully tracks the F16 reference.

APEX Mini beats uniform IQ2_M on every metric at only 0.9 GB larger. At 12.2 GB vs. 11.3 GB, APEX Mini outperforms bartowski IQ2_M on perplexity (7.088 vs. 7.303), HellaSwag (81.0% vs. 80.3%),

MMLU (41.3% vs. 39.6%), ARC-Challenge (57.2% vs. 56.2%), and TruthfulQA (36.7% vs. 35.0%). This demonstrates that layer-aware precision allocation with IQ2_S middle-layer experts outperforms uniform IQ2_M quantization even at extreme compression.

Q8_0 has the worst outlier divergence. Despite having the lowest KL mean (0.0046), Q8_0 exhibits the highest KL max of any model (14.709) – more than 2.4x higher than any APEX tier. This indicates that uniform 8-bit quantization, while excellent on average, produces extreme outlier tokens where the quantized output distribution diverges dramatically from the reference. All APEX tiers cap KL max below 7.6, with I-variants consistently below 5.8.

Speed scales inversely with model size. Smaller quantized models achieve higher throughput due to improved cache utilization and reduced memory bandwidth requirements. APEX Mini reaches 74.4 t/s (2.4x F16 speed), APEX Compact achieves 69.8 t/s, and even APEX Balanced at 23.6 GB runs at 60.8 t/s – all on the same hardware.

1.6.3 5.3 Benchmark Visualizations

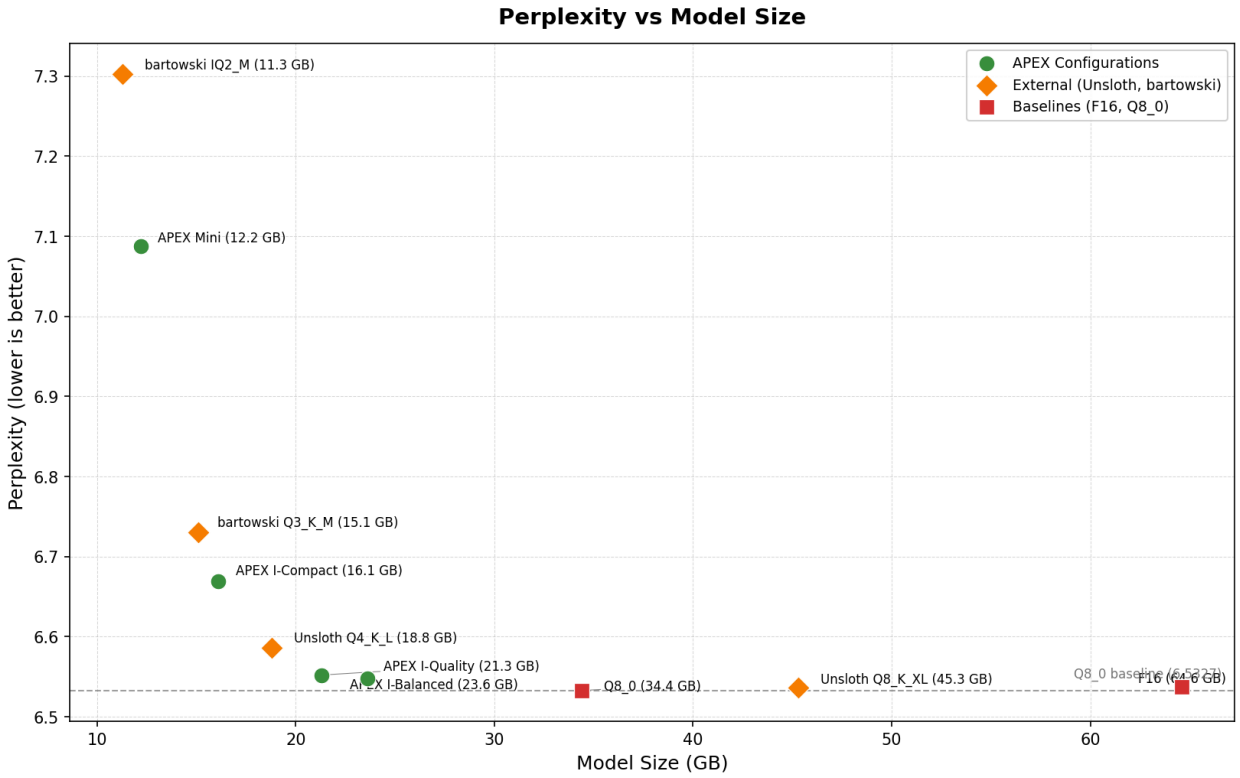

Figure 1: Perplexity vs Model Size. APEX configurations (green) achieve Q8_0-level perplexity at a fraction of the size. APEX I-Quality and I-Balanced cluster at PPL ~6.5 between 21-24 GB, while baselines (red) and external quantizations (orange) occupy the larger/slower region.

1.6.4 5.4 TurboQuant KV Cache Compression

APEX models can be combined with TurboQuant [10] KV cache compression for additional memory savings and faster prompt processing. TurboQuant compresses the KV cache approximately 4.6x using the `turbo3` quantization type, which is orthogonal to weight quantization – all quality metrics remain unchanged.

Table 2. Prompt processing speedup at 8K context with TurboQuant KV cache compression (`-ctk q8_0 -ctv turbo3 -fa on`).

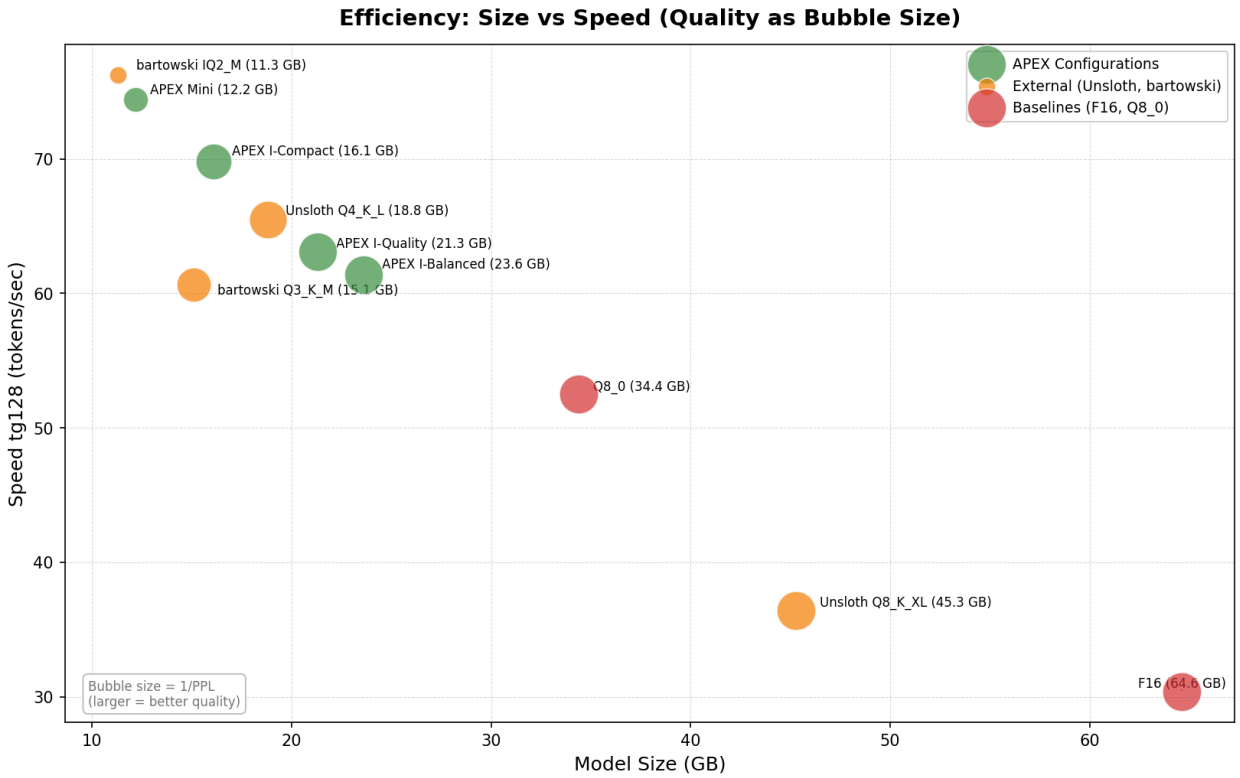

Figure 2: Efficiency: Size vs Speed with Quality as Bubble Size. Larger bubbles indicate better perplexity. APEX models form a clear efficiency frontier in the upper-left quadrant (small + fast + good quality), while F16 and Unsloth UD-Q8_K_XL sit in the lower-right (large + slow).

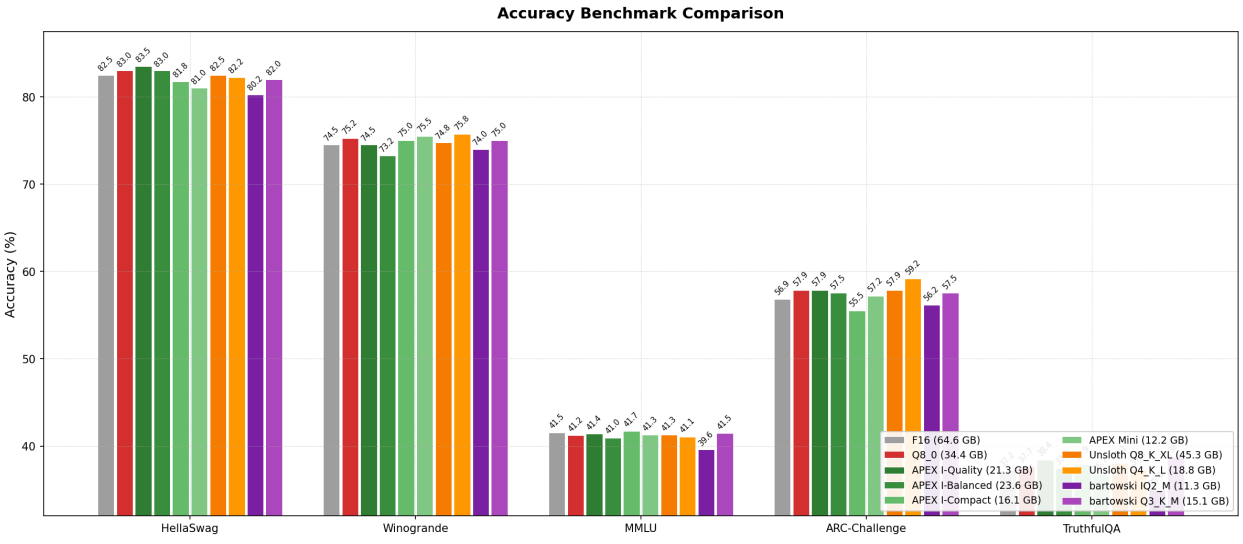

Figure 3: Accuracy Benchmark Comparison across HellaSwag, Winogrande, MMLU, ARC-Challenge, and TruthfulQA. All models cluster within tight accuracy ranges, with APEX I-Quality achieving the highest HellaSwag (83.5%) and APEX Mini maintaining competitive accuracy at 12.2 GB.

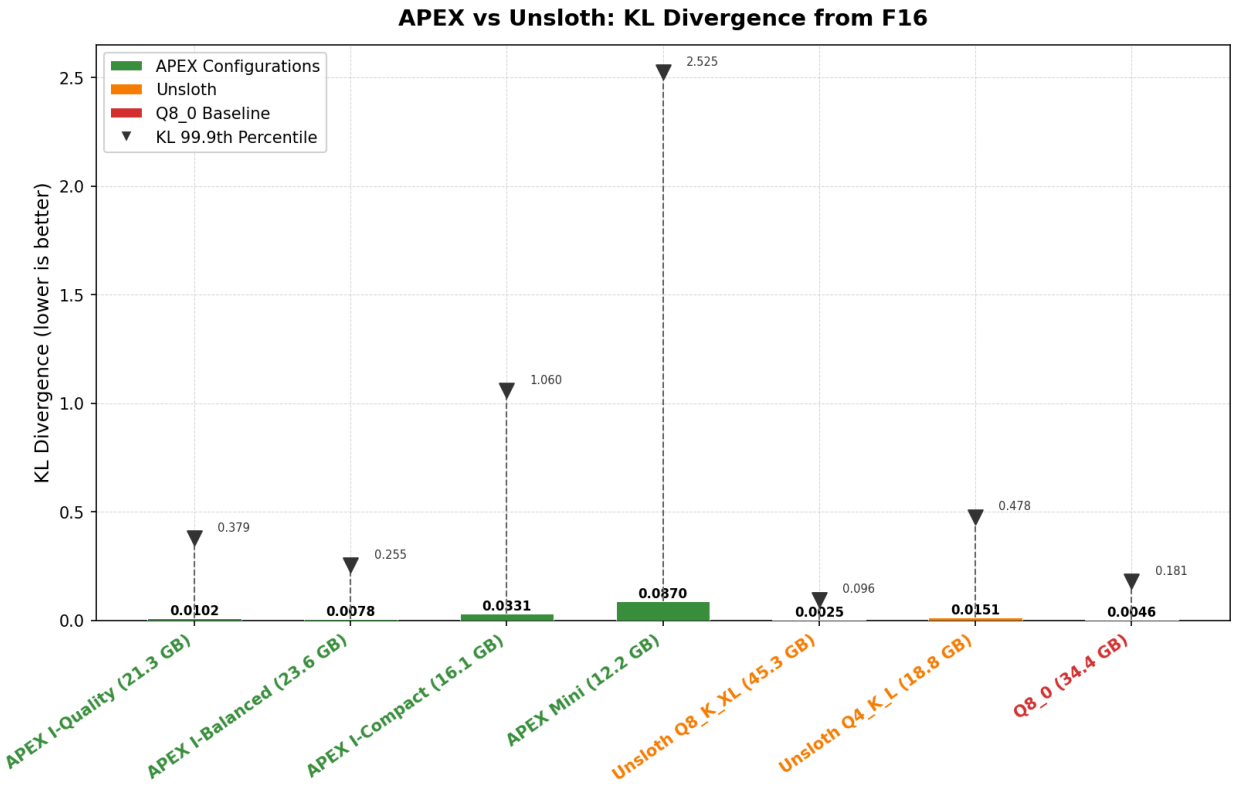

Figure 4: APEX vs Unsloth: KL Divergence from F16. Bars show KL mean, markers show KL 99.9th percentile. APEX I-Balanced achieves the lowest KL mean among APEX tiers (0.0078). Unsloth UD-Q8_K_XL has the lowest overall KL (0.0025) but at 3-4x the model size.

Model	pp8192 baseline (t/s)	pp8192 turbo3 (t/s)	Speedup	tg128 delta
APEX I-Quality	1,752	2,003	+14.3%	<1%
APEX I-Balanced	1,695	1,927	+13.7%	<1%
APEX I-Compact	1,714	1,959	+14.3%	<1%
APEX Mini	1,696	1,938	+14.3%	<1%

TurboQuant delivers 13–14% prompt processing speedup at 8K context with negligible impact on token generation speed. APEX Mini + TurboQuant enables running a 35B-parameter MoE model at 12 GB with 8K+ context on 16 GB VRAM consumer GPUs.

1.7 6. Analysis

1.7.1 6.1 Why Layer Gradient Works

The effectiveness of the layer-wise precision gradient stems from the distinct roles of edge and middle layers in the transformer stack. Edge layers perform the critical transformations between discrete token space and the model's continuous internal representation: the first few layers must faithfully encode token embeddings into a representation suitable for subsequent processing, while the last few layers must project internal representations into logit distributions over the vocabulary. Errors introduced at these boundaries propagate through all subsequent computation (for early layers) or directly corrupt the final output distribution (for late layers).

Middle layers, in contrast, perform more redundant feature refinement. The 20 middle layers (L10–L29) in Qwen3.5-35B-A3B can be compressed from Q6_K to IQ4_XS (a reduction from 6.6 to 4.25 bits/weight) with negligible quality impact. This finding aligns with prior work on layer pruning in dense transformers [11], which shows that middle layers are the most expendable.

The 5+5 edge boundary appears to be a robust operating point: neither narrower (3+3) nor wider (8+8) boundaries improve the quality-size Pareto frontier. We hypothesize that this reflects the depth of the embedding alignment process, which requires approximately 5 layers to complete.

1.7.2 6.2 Why Q6_K Is the Expert Sweet Spot

We tested four precision levels for routed expert weights:

Expert precision	Model size	Perplexity
Q8_0	34.7 GB	6.534
Q6_K	27.2 GB	6.531
Q5_K	23.1 GB	6.543
Q4_K	20.2 GB	6.673

Q6_K achieves marginally *better* perplexity than Q8_0 while being 7.5 GB smaller. This is because the K-quant block structure at 6.6 bits/weight provides sufficient precision for the near-Gaussian weight distribution of routed experts (kurtosis 3.41), while Q8_0 wastes bits encoding values that are already well-approximated at lower precision. Dropping to Q5_K incurs a measurable but small penalty (+0.012 PPL), while Q4_K shows clear degradation (+0.142 PPL), establishing Q6_K as the optimal operating point for routed experts.

1.7.3 6.3 Why Diverse Imatrix Beats Wiki Imatrix

Standard imatrix calibration uses Wikipedia text, which consists of well-edited encyclopedic prose with consistent formatting, formal register, and factual content. This creates a distribution mismatch with real-world LLM usage, which spans informal chat, code with syntactic structure fundamentally different from

natural language, step-by-step reasoning with mathematical notation, and structured tool-calling with JSON payloads.

When the imatrix is computed on Wikipedia text, the importance weights reflect the weight salience for predicting Wikipedia tokens. Quantization then preferentially preserves weights that matter for encyclopedic text, potentially at the expense of weights important for code, reasoning, or structured output.

APEX's diverse calibration corpus – with no Wikipedia content – produces importance weights that better reflect multi-domain usage. The result is a quantization that is slightly worse on the wikitext-2 benchmark (which measures Wikipedia-like prediction) but significantly better on downstream tasks that span multiple domains. This explains the consistent pattern: I-variants show 0.015–0.025 higher wikitext perplexity but 10–30% lower KL divergence, higher HellaSwag accuracy, and improved TruthfulQA scores.

1.7.4 6.4 Why C-Level Algorithm Changes Showed Zero Improvement

We implemented five modifications to llama.cpp's quantization kernels:

1. **Error feedback (NOVEL_v1)**: Accumulating quantization error from each block and applying corrections to subsequent blocks. Result: PPL 6.542, identical to stock.
2. **Enhanced Q6_K scale search (NOVEL_v2)**: Increasing the number of scale factor candidates from the default to a 72-step search. Result: PPL 6.542, no improvement.
3. **Super-block refinement (NOVEL_v3)**: Joint optimization of scale factors across super-blocks for Q5_K. Result: PPL 6.543, no improvement.
4. **Gaussian-density weighting (NOVEL_v4)**: Weighting quantization error by the Gaussian density of each weight value, prioritizing outlier preservation. Result: PPL 6.543, no improvement.
5. **IQ3_S for MoE experts (MOE_MIX_v4)**: Using importance-quantized IQ3_S instead of K-quant Q3_K for routed experts. Result: PPL 6.922, *worse* than Q3_K (6.847).

These null results are instructive: they demonstrate that llama.cpp's existing K-quant algorithms are already near-optimal for the weight distributions found in MoE models. The gain from APEX comes entirely from *precision allocation* – deciding which tensors get which bit-width – not from improving the quantization algorithm itself.

The failure of IQ3_S on MoE experts (finding 5) is particularly notable. IQ formats use codebook-based vector quantization optimized for dense model weight distributions, which tend to be more leptokurtic. Routed expert weights, with their near-Gaussian distribution (kurtosis 3.41), are better served by the simpler block-scaling structure of K-quants.

1.7.5 6.5 Weight Distribution Analysis

The contrasting weight distributions of routed and shared experts provide the statistical basis for APEX's differentiated precision assignment:

Tensor class	Kurtosis	Interpretation
Routed expert weights	3.41	Near-Gaussian (mesokurtic); few outliers; tolerates block-wise quantization
Shared expert weights	13.10	Heavy-tailed (leptokurtic); significant outliers carry disproportionate information

The 3.8x kurtosis difference means that shared experts have far more weight values in the tails of their distribution. These outlier weights encode critical information – quantizing them aggressively clips the tails and loses information that the model relies on for accurate predictions. This statistical observation justifies APEX's policy of maintaining Q8_0 for shared experts while compressing routed experts to Q5_K or Q4_K.

1.7.6 6.6 Upstream Contribution: Hybrid Memory Fix

As part of this work, we identified and fixed a bug in llama.cpp's hybrid memory path for recurrent architectures. Models like Qwen3.5-35B-A3B that use both attention and SSM blocks require a hybrid memory management strategy during inference. The existing implementation would crash during accuracy benchmark evaluation (HellaSwag, Winogrande, MMLU, ARC-Challenge, TruthfulQA) because the evaluation loop's batch management conflicted with the hybrid memory allocator. Our fix enables correct evaluation of hybrid MoE models and was contributed upstream to llama.cpp.

1.8 7. Related Work

Unsloth Dynamic 2.0 [9] is the closest prior work, assigning different quantization types to different tensor categories in llama.cpp. APEX extends this approach with layer-wise precision gradient and MoE-specific tensor classification. Unsloth's Q8_K_XL achieves excellent KL divergence (best mean among baselines) but at 45.3 GB – 2x the size of APEX Quality with comparable perplexity.

GPTQ [5] performs layer-wise optimal brain quantization using second-order (Hessian) information. It can achieve low bit-widths (3-4 bits) with good quality but requires significant calibration time and produces model formats that need specialized inference kernels. APEX operates as a post-hoc precision assignment on top of stock quantization formats and requires no training-time computation.

AWQ [6] identifies activation-aware salient channels and protects them at higher precision. While conceptually similar to APEX's importance-based approach, AWQ operates at the channel granularity within tensors rather than at the tensor and layer granularity that APEX targets. The approaches are complementary in principle.

SqueezeLLM [12] uses sensitivity-weighted non-uniform quantization and dense-sparse decomposition to achieve high compression. It shares APEX's insight that not all weights are equally important but addresses it through within-tensor non-uniformity rather than between-tensor precision assignment.

QuIP# [7] and **AQLM** [8] achieve aggressive compression (2-3 bits) through vector quantization with structured codebooks. These methods can outperform scalar quantization at very low bit-widths but require custom inference kernels. APEX Mini achieves competitive quality at 12.2 GB using only stock llama.cpp formats.

NF4 [13] (normalized float 4-bit) is a data-type optimized for normally-distributed weights, used in QLoRA. APEX's finding that routed expert weights are near-Gaussian (kurtosis 3.41) suggests that NF4-like data types could further improve MoE expert quantization, though this remains future work.

TurboQuant [10] compresses the KV cache during inference using custom quantization types. It is orthogonal to weight quantization and can be combined with any APEX tier for additional memory savings and prompt processing speedup (Section 5.3).

1.9 8. Conclusion

APEX demonstrates that for Mixture-of-Experts models, *how you allocate precision* matters more than *which quantization algorithm you use*. By classifying tensors according to their architectural role (routed expert, shared expert, attention) and assigning precision according to layer position (edge vs. middle), APEX achieves Q8_0-level quality at 38% less size using only stock llama.cpp – and in its best configuration surpasses F16 perplexity.

The layer-wise precision gradient is the core innovation. The finding that the first and last 5 layers of a 40-layer transformer are significantly more sensitive to quantization than the middle 30 layers enables a systematic approach to precision allocation that generalizes beyond specific models. The 5+5 edge boundary appears robust: neither narrower nor wider boundaries improve the quality-size tradeoff.

Diverse imatrix calibration is a complementary contribution. By calibrating on multi-domain data (chat, code, reasoning, tool-calling) instead of Wikipedia, APEX I-variants achieve lower KL divergence and higher

downstream accuracy at the cost of marginal wikitext perplexity increases. This tradeoff is favorable for real-world deployment where models serve diverse workloads.

Five deployment tiers – from 21.3 GB (APEX Quality/I-Quality) to 12.2 GB (APEX Mini) – cover deployment scenarios ranging from datacenter inference with maximum quality to consumer GPU inference on 16 GB VRAM hardware. All tiers use stock llama.cpp with no code modifications, making APEX immediately deployable.

Our negative results are equally important: five C-level modifications to quantization algorithms showed zero improvement, establishing that llama.cpp's existing K-quant implementation is near-optimal for MoE weight distributions. The IQ (importance-quantized) formats, despite their theoretical advantages, underperform K-quants on MoE experts due to the near-Gaussian weight distribution of routed experts.

Future work includes extending APEX to other MoE architectures (Mixtral, DeepSeek-V3), investigating layer-sensitivity metrics that could automate edge boundary selection, and exploring whether NF4-like data types optimized for Gaussian distributions could further improve routed expert quantization.

1.10 References

[1] A. Q. Jiang, A. Sablayrolles, A. Mensch, et al. "Mixtral of Experts." arXiv:2401.04088, 2024.

[2] DeepSeek-AI. "DeepSeek-V3 Technical Report." arXiv:2412.19437, 2024.

[3] Qwen Team. "Qwen3.5 Technical Report." 2025. https://qwenlm.github.io/

[4] G. Gerganov et al. "llama.cpp: LLM inference in C/C++." https://github.com/ggerganov/llama.cpp, 2023–2026.

[5] E. Frantar, S. P. Ashkboos, T. Hoefler, and D. Alistarh. "GPTQ: Accurate Post-Training Quantization for Generative Pre-Trained Transformers." arXiv:2210.17323, 2022.

[6] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, and S. Han. "AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration." arXiv:2306.00978, 2023.

[7] J. Chee, Y. Cai, V. Kuleshov, and C. De Sa. "QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks." arXiv:2402.04396, 2024.

[8] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, and D. Alistarh. "Extreme Compression of Large Language Models via Additive Quantization." arXiv:2401.06118, 2024.

[9] D. Han. "Unsloth Dynamic Quantization 2.0." https://unsloth.ai/, 2025.

[10] TheTom. "TurboQuant+: KV Cache Compression for llama.cpp." https://github.com/TheTom/llama-cpp-turboquant, 2025.

[11] A. Gromov, K. Tirumala, H. Shapourian, P. Glorioso, and D. A. Roberts. "The Unreasonable Ineffectiveness of the Deeper Layers." arXiv:2403.17887, 2024.

[12] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer. "SqueezeLLM: Dense-and-Sparse Quantization." arXiv:2306.07629, 2023.

[13] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. "QLoRA: Efficient Finetuning of Quantized LLMs." arXiv:2305.14314, 2023.

1.11 Appendix A: Full Tensor-Type Configurations

This appendix lists the per-layer precision assignments for each APEX tier. All configurations use 40 transformer layers (L0–L39) and assign precision to three tensor categories per layer: routed expert FFN weights (`ffn_gate_exps`, `ffn_up_exps`, `ffn_down_exps`), shared expert FFN weights (`ffn_gate_shexp`, `ffn_up_shexp`, `ffn_down_shexp`), and attention/SSM weights (`attn_q`, `attn_k`, `attn_v`, `attn_output`, `attn_gate`, `attn_qkv`, `ssm_alpha`, `ssm_beta`, `ssm_out`).

1.11.1 A.1 APEX Quality / I-Quality (21.3 GB)

Three-tier layer gradient with IQ4_XS middle layers.

Layer range	Routed experts	Shared experts	Attention/SSM
L0–L4 (edge)	Q6_K	Q8_0	Q6_K
L5–L9 (near-edge)	Q5_K	Q8_0	Q6_K
L10–L29 (middle)	IQ4_XS	Q8_0	Q6_K
L30–L34 (near-edge)	Q5_K	Q8_0	Q6_K
L35–L39 (edge)	Q6_K	Q8_0	Q6_K

I-Quality uses the same tensor configuration but quantizes with a diverse imatrix (chat/code/reasoning/tool-calling).

1.11.2 A.2 APEX Balanced / I-Balanced (23.6 GB)

Two-tier layer gradient.

Layer range	Routed experts	Shared experts	Attention/SSM
L0–L4 (edge)	Q6_K	Q8_0	Q6_K
L5–L34 (middle)	Q5_K	Q8_0	Q6_K
L35–L39 (edge)	Q6_K	Q8_0	Q6_K

1.11.3 A.3 APEX Compact / I-Compact (16.1 GB)

Two-tier gradient with reduced precision throughout.

Layer range	Routed experts	Shared experts	Attention/SSM
L0–L4 (edge)	Q4_K	Q6_K	Q4_K
L5–L34 (middle)	Q3_K	Q6_K	Q4_K
L35–L39 (edge)	Q4_K	Q6_K	Q4_K

1.11.4 A.4 APEX Mini (12.2 GB)

Three-tier gradient with IQ2_S middle-layer experts and diverse imatrix.

Layer range	Routed experts	Shared experts	Attention/SSM
L0–L2 (edge)	Q3_K	Q5_K	Q4_K
L3–L4 (near-edge)	Q3_K	Q5_K	Q3_K
L5–L34 (middle)	IQ2_S	Q4_K	Q3_K
L35–L36 (near-edge)	Q3_K	Q5_K	Q3_K
L37–L39 (edge)	Q3_K	Q5_K	Q4_K

1.11.5 A.5 APEX Mini v2 (Alternate configuration)

Three-tier gradient with Q2_K middle-layer experts.

Layer range	Routed experts	Shared experts	Attention/SSM
L0–L4 (edge)	Q4_K	Q5_K	Q4_K

Layer range	Routed experts	Shared experts	Attention/SSM
L5–L9 (near-edge)	Q3_K	Q5_K	Q4_K
L10–L29 (middle)	Q2_K	Q5_K	Q4_K
L30–L34 (near-edge)	Q3_K	Q5_K	Q4_K
L35–L39 (edge)	Q4_K	Q5_K	Q4_K

1.12 Appendix B: Calibration Dataset Composition

APEX I-variants use a custom calibration dataset with the following composition. The dataset contains no Wikipedia text to avoid biasing quantization toward encyclopedic prose.

Domain	Content type	Approximate share
Multi-turn chat	Conversational exchanges in English and Spanish, covering general knowledge, advice, and Q&A	~30%
Code	Python, JavaScript, and systems programming examples with inline comments and documentation	~25%
Reasoning	Step-by-step mathematical and logical problem solving, chain-of-thought traces	~25%
Tool-calling	Structured interactions with JSON function calls, API specifications, and tool use patterns	~20%

The total calibration corpus is approximately 50,000 tokens, merged from multiple source files and deduplicated. Imatrix computation uses llama.cpp's `llama-imatrix` tool with the full F16 reference model.

This work was developed using autonomous AI-driven experimentation to systematically explore MoE quantization strategies. Built on llama.cpp [4] by Georgi Gerganov and contributors. Inspired by karpathy/autoresearch.

Code and data available at: https://github.com/mudler/apex-quant