1.4 The Accuracy Cost of Quantization

Chapter 1.2 laid out the quantization recipes Intel NPU supports: INT8-sym, INT4-sym group-128 or channel-wise, NF4 on Lunar Lake, FP8 on Panther Lake. The hardware story ended there. This section is the missing other half — what those recipes actually cost you in model quality, how to measure it, and when the cost stops being acceptable.

It matters because the gap between "this compiles and runs" and "this works for users" is wider than the OpenVINO docs suggest. Quantization is never free. You're trading bits of weight precision for bandwidth and memory headroom, and somewhere on the spectrum from FP16 down to INT4 the model starts being meaningfully worse at the thing you're paying it to do. The job is to find where that line is for your specific workload, not to assume Intel's validated recipes are quality-validated for every task.

Why Quantization Isn't Free

A 4-bit weight has 16 distinct values. The full-precision FP16 weight it replaces has roughly 65,000. Group-wise quantization mitigates this by sharing a scale factor across 128 weights — so the effective dynamic range per group is closer to FP16's, but every weight in the group still has to round to one of 16 levels relative to that scale. Channel-wise quantization is more aggressive: one scale per output channel, hundreds or thousands of weights sharing it.

For weights with a broad distribution and a few outliers (which is what most transformer layers have), this rounding error compounds layer by layer. By the time a token has traversed 24 transformer blocks, the accumulated drift is large enough to change which next-token probability wins. Sometimes that's invisible — the model still says something reasonable. Sometimes it cascades into hallucinated facts, broken JSON, or a translation that means something subtly different from what the source said.

The intuition that matters: **decode is more sensitive to quantization than prefill**. Prefill processes the entire prompt in one parallel pass; small errors get averaged across many tokens. Decode generates one token at a time, with each new token's logits depending on every prior step's KV state. An error introduced at step 5 propagates through step 6, 7, 8 — and the model can't "self-correct" because greedy decoding (the only mode on NPU `LLMPipeline`) commits to whichever token won the contaminated argmax. This is why aggressive quantization tends to break agents specifically: agents do long decodes, and the drift accumulates.

The Standard Intrinsic Metric: Perplexity

Perplexity measures how surprised a model is by held-out text. Lower is better. A model that quantizes well sees its perplexity rise by 1–3% relative to the FP16 baseline; a model that quantizes badly sees 10%+ rises and tangible output degradation.

Perplexity is computed on a fixed corpus (WikiText-2 is conventional) by running the model forward and averaging the negative log-likelihood of each token in context. The mechanics:

```
# Sketch — use lm-evaluation-harness in practice
import math
from transformers import AutoTokenizer
import openvino_genai as ov_genai

tokenizer = AutoTokenizer.from_pretrained("path/to/model")
pipe = ov_genai.LLMPipeline("path/to/openvino_ir", "NPU")

total_nll = 0.0
total_tokens = 0
for text in wikitext_test_corpus:
    tokens = tokenizer.encode(text)
    # Run the model in teacher-forced mode; sum -log P(token_i | tokens_<i)
    nll, n = score_sequence(pipe, tokens)  # implementation-specific
    total_nll += nll
    total_tokens += n

ppl = math.exp(total_nll / total_tokens)
```

In practice you use `lm-evaluation-harness` (`lm-eval --tasks wikitext`) against the OpenVINO model and against the FP16 baseline; the comparison is what tells you anything. A standalone perplexity of 7.2 means nothing without context. A perplexity that jumped from 6.8 (FP16) to 7.5 (INT4-sym group-128) means a 10% increase — borderline acceptable for chat, probably noticeable for translation.

Perplexity is a useful canary, not a verdict. A model can have stable perplexity but fail on the specific structured-output task your agent depends on. Always pair it with task-level evaluation.

Downstream Task Evaluation

The benchmarks that matter for agents:

- **MMLU** (Massive Multitask Language Understanding) — multiple-choice across 57 academic subjects. Tests knowledge retention; sensitive to quantization in mid-range models (3B–8B). Expect 1–3 percentage points of accuracy loss at INT4-sym group-128.
- **IFEval** — instruction-following evaluation. Measures whether the model obeys constraints like "respond in JSON" or "answer in exactly three sentences." **This is the test most directly relevant to agent reliability.** Quantization-sensitive in a non-linear way: models often pass at INT8, scrape by at INT4 group-128, fail at INT4 channel-wise.
- **HumanEval** / **MBPP** — Python code generation. If your agent writes code, run these. Aggressive quantization tends to break syntax (extra parentheses, missing colons) before it breaks logic.
- **MT-Bench** — multi-turn conversation quality with LLM-as-judge scoring. Most expensive to run; most representative of chat use cases.
- **BLEU / chrF / COMET for translation** — if you're following the M2M-100 thread of this book, these are the metrics that matter. Quantization can drop BLEU by 1–2 points on common language pairs and 3–5 points on low-resource pairs.

Run every benchmark twice: once against the FP16 baseline you're shipping with, once against the quantized OpenVINO export. Report the delta, not the absolute number. The delta is what tells you whether the quantization recipe is safe.

Empirical Findings, Roughly

The book can't promise specific numbers for every model — they vary too much — but the shape of the curve generalizes. For decoder-only LLMs in the 3B–8B range:

Recipe	Perplexity delta vs FP16	MMLU delta	IFEval delta	Typical verdict
INT8 weight-only sym	+0.5 to +1.5%	-0 to -1 pp	-0 to -1 pp	Safe; ship it
INT4-sym group-128	+2 to +5%	-1 to -3 pp	-1 to -4 pp	Usually acceptable; verify on your task
INT4-sym channel-wise	+5 to +10%	-2 to -5 pp	-3 to -8 pp	Acceptable only for large models (>10B)
NF4 channel-wise	+2 to +4%	-1 to -2 pp	-1 to -3 pp	Close to INT8 quality; Lunar Lake+ only
FP8 (E4M3)	+0.5 to +1.5%	-0 to -1 pp	-0 to -1 pp	Close to FP16 quality; Panther Lake+ only

The values are illustrative ranges. Your model will land somewhere in them, and where it lands depends on training data, model family, and the specifics of NNCF's calibration. Don't quote this table to anyone as if it were measured on their model. It isn't.

For **encoder-decoder seq2seq models** like M2M-100, the numbers tend to be worse than decoder-only at the same recipe. Two reasons. First, the encoder and decoder accumulate errors independently, and the cross-attention exposes both. Second, translation is a "every token matters" task: a mistranslated noun isn't a soft degradation, it's a wrong answer. Where decoder-only LLMs can lose 2 MMLU points without anyone noticing, a seq2seq translator that loses 2 BLEU is detectably worse to a user. Test more aggressively; consider INT8 weights as your floor rather than INT4 if quality matters.

The Specific Failure Modes for Agents

Agents fail at quantization in characteristic ways:

Structured output breaks first. The model that produced valid JSON at FP16 starts emitting trailing commas, unbalanced braces, or extra commentary outside the fence. This is because JSON is a low-probability tail of the model's output distribution; small logit perturbations are enough to push it into the higher-probability "natural language" basin. Mitigation: constrained decoding (Chapter 3.4), or step back to a less aggressive quantization.

Long-decode coherence degrades. The first 50 tokens of an output look fine; tokens 200–500 drift into repetition, off-topic content, or factual confusion. The drift is the KV-cache-accumulating-quantization-error effect. Mitigation: tighter context windows, shorter task decomposition, or larger model at less aggressive quantization.

Instruction following weakens. "Answer in exactly three sentences" becomes "answer in roughly three or four sentences." "Output only the translated text" becomes "Here is the translation: [translated text]." This shows up as IFEval drops and is one of the most common silent failures.

Multi-step reasoning hallucinates more. Chain-of-thought arguments stay grammatical but become factually wrong, or arrive at correct conclusions via incorrect intermediate steps. Particularly bad for plan-then-execute agents where the plan depends on accurate reasoning at step 1.

Rare-token tasks break. Anything involving uncommon vocabulary (medical terms, proper nouns, code identifiers) degrades faster than common-prose tasks. The 4-bit weight space simply doesn't have the precision to discriminate among rare-token logits as cleanly as FP16 does.

Why Asymmetric Quantization Crashes the NPU LLM Path

Chapter 1.2 mentioned the rule — `--sym --ratio 1.0` for NPU LLMs, asymmetric quantization crashes the compile path — without explaining why. The reason is worth understanding because it

tells you something about what the NPU compiler is doing.

Symmetric quantization represents the weight range as `[-scale × 7, +scale × 7]` for 4-bit (or analogous ranges for other bit widths); the zero-point is fixed at 0 and not encoded per-weight. Asymmetric quantization adds a per-tensor or per-group zero-point shift: `[-scale × 7 + zp, +scale × 7 + zp]`, encoding both scale and zero-point.

The NPU compiler's matmul kernels assume the symmetric layout. The MAC arrays are wired to multiply against a fixed offset of zero. Asymmetric weights would require a per-input-channel adjustment that doesn't exist in the kernel templates Intel ships. The result isn't a quality degradation; it's a compile failure, often with an opaque error message about kernel selection.

This is changing — Intel has talked about asymmetric INT4 support in future releases — but as of OpenVINO 2026.1, sticking to symmetric is mandatory for LLM workloads on NPU.

When Quantization Is Breaking Your Model

Specific signs to watch for:

Validation perplexity rises >5%. Stop and reconsider. Either the recipe is too aggressive, the calibration dataset doesn't match your domain, or there's a layer that quantizes badly and needs to be excluded.

Greedy decoding produces obviously different output than the FP16 baseline on the same prompt. Greedy is deterministic; if FP16 and INT4 give different answers, the rounding has shifted the argmax somewhere material. Run the same prompt 5–10 times; pattern-match the differences.

Specific tokens become impossible. Hashes, code identifiers, low-frequency vocabulary words may simply never be selected because their FP16 logit advantage gets rounded away. Easy to detect by checking the top-10 candidate tokens at known critical positions.

Outputs get longer or shorter systematically. The end-of-sequence token's logit shifts under quantization. If outputs are systematically truncated, the EOS token is being selected too early; if they ramble past the natural endpoint, it's being selected too late.

NNCF's Accuracy-Aware Workflow

When you have a quality budget and a calibration dataset, NNCF offers a workflow that automates layer-by-layer precision selection. The mechanics, roughly:

1. Run the FP16 model on your calibration dataset; record per-layer activation statistics.
2. Try quantizing each layer in isolation; measure perplexity / task accuracy delta.
3. Identify which layers contribute disproportionately to quality loss.
4. Apply less aggressive precision (or skip quantization) on the problematic layers; keep the rest aggressive.
5. Iterate until quality is within budget.

In Optimum-Intel:

```
# Sketch — actual flags vary by version
optimum-cli export openvino \
  --model facebook/m2m100_1.2B \
  --task text2text-generation-with-past \
  --weight-format int4 --sym --ratio 0.8 --group-size 128 \
  --quantization-config '{"sensitivity_metric": "weight_quantization_error"}' \
  m2m100_1.2B_ov_int4_mixed
```

The `--ratio 0.8` says "quantize 80% of layers to INT4; keep the most sensitive 20% at INT8." NNCF picks the layers based on the sensitivity metric. This is a real lever for tuning quality-vs-size when uniform quantization is too aggressive. It also costs you some compile complexity and a longer export step.

What This Section Bought You

You should now understand:

- **Quantization isn't free** — somewhere on the precision spectrum, model quality starts degrading materially
- **Decode is more quantization-sensitive than prefill** — agents (which do long decodes) feel quantization more than one-shot inference
- **Perplexity is a canary, not a verdict** — pair it with task-level benchmarks (MMLU, IFEval, HumanEval, MT-Bench, BLEU)
- **Empirical ranges**: INT8 ~1% degradation, INT4 group-128 ~3% on decoder-only LLMs; encoder-decoder is worse
- **Agent failure modes**: structured output breaks first, long-decode coherence drifts, instruction following weakens, rare tokens vanish
- **Asymmetric quantization crashes the NPU LLM compile path** because the MAC kernels assume symmetric layout
- **NNCF accuracy-aware workflow** lets you mix precisions per layer when uniform quantization hurts too much
- **Always compare to your FP16 baseline** — absolute quantized numbers mean nothing in isolation

The next section steps back to the bandwidth ceiling we established and asks whether there's any way to dodge it — specifically, whether speculative decoding (now available on NPU) can buy us back the throughput the LPDDR5X bus refuses to give.

Previous: *1.3 Latency, Throughput, and Hardware-Aware Patterns* **Next:** *1.5 Speculative Decoding*

Revision #1
Created 2026-05-12 19:34:49 UTC by Admin
Updated 2026-05-12 19:34:49 UTC by Admin