

StaffML: A Physics-Grounded Interview Question Bank for Machine Learning Systems Engineers

Vijay Janapa Reddi
Harvard University

staffml.ai

Abstract

As large language models commoditize algorithmic coding, the defining skill for Staff-level ML systems engineers is *mechanical sympathy*, the ability to reason quantitatively about how software interacts with physical hardware. No structured assessment resource exists for this skill, so candidates overprepare on coding puzzles, hiring teams improvise system-design loops, and educators have no shared map of what to teach. We present STAFFML, a bank of 9,521 interview questions that closes this gap. Each question is a paragraph-sized scenario with a worked back-of-the-envelope (“napkin math”) derivation grounded in real accelerator specifications (H100, MI300X, Orin, Cortex-M4), so every answer is a number the candidate must derive rather than retrieve. Every question is classified on four independent axes. The *topic* axis covers 87 concepts in a typed prerequisite graph. The *zone* axis names a cognitive mode drawn from a four-skill competency model whose pairwise intersections yield 11 zones such as diagnosis, specification, and fluency. The *track* axis fixes the physics regime across cloud, edge, mobile, and TinyML. The *level* axis encodes Bloom’s depth and aligns with industry engineering ladders. The bank ships with the infrastructure to keep that classification trustworthy at scale, including a LinkML schema with validate-at-write Pydantic constraints, three-stage deduplication, multi-model math verification, and a 26-check invariant suite that rejects inconsistent items before commit. The contribution is this complete assessment apparatus, which gives candidates calibrated practice, hiring teams a shared debrief vocabulary, and educators a coverage map.

Part of the *Machine Learning Systems* book ecosystem; book context: <https://mlsysbook.ai/staffml/>.

1 Introduction

The ML systems engineer sits at the intersection of hardware and software. Unlike ML researchers who design model architectures or software engineers who write application code, ML systems engineers reason about the physical constraints of deploying machine learning at scale: memory bandwidth limits that determine whether a workload is compute-bound or memory-bound, thermal envelopes that cap sustained throughput on edge devices, network congestion that dominates training time in distributed clusters, and the economics of accelerator fleets where a single misconfigured batch size can waste thousands of dollars per hour.

This is what Martin Thompson calls *mechanical sympathy* (Thompson, 2011), a term borrowed from racing driver Jackie Stewart’s philosophy that a driver performs best when they understand what the car is doing underneath them. A Formula 1 driver who understands the engine does not merely drive faster; they drive differently, making decisions that are physically impossible to arrive at through pure algorithmic thinking. Similarly, an ML systems engineer who understands that autoregressive decode reads the full 140 GB model weights for every single token, achieving an arithmetic intensity of just 1 FLOP/Byte, far below the hardware’s 148 FLOP/Byte ridge point, does not merely optimize code; they architect systems that batch requests to amortize the bandwidth cost. (Throughout this paper, *Staff+* follows industry shorthand for Staff-and-above individual-contributor levels such as Senior Staff and Principal; *Staff-level* names the same senior band when we discuss skills, roles, or interviews.)

As large language models commoditize algorithmic coding (OpenAI, 2023), the traditional coding interview has lost its predictive power for Staff+ hiring. You cannot prompt your way out of a memory

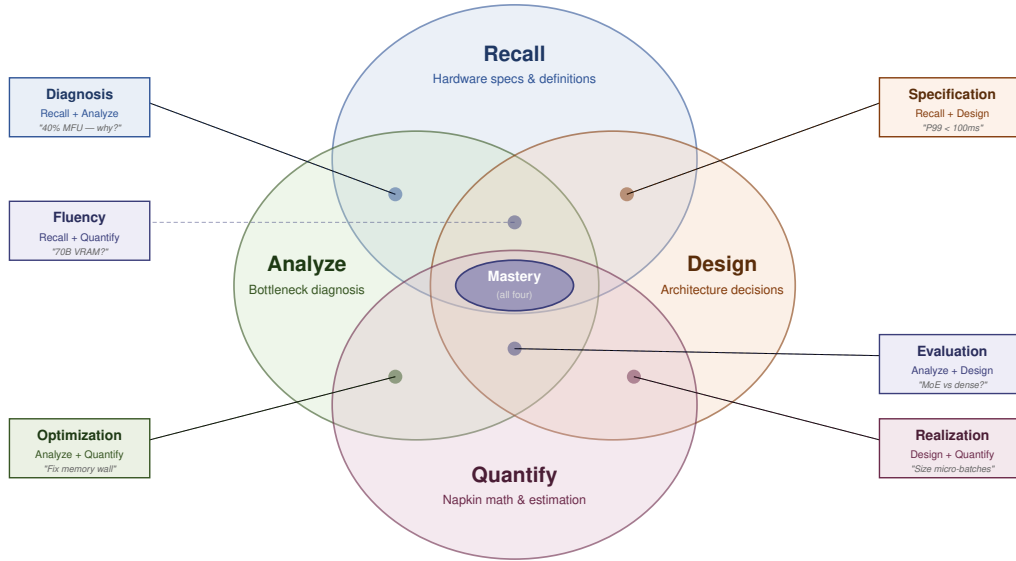


Figure 1. The STAFFML four-skill competency model. Four fundamental skills (Recall, Analyze, Design, Quantify) overlap to produce six compound zones and one central mastery zone. The four-circle Venn structure is borrowed from the *ikigai* life-philosophy diagram; the content (recall/analyze/design/quantify) is our own decomposition of ML systems expertise. Each question targets exactly one zone, which determines how a topic is assessed. The compound zones capture the cognitive acts that distinguish Staff-level engineers: diagnosing failures ($\text{Recall} \cap \text{Analyze}$), translating specifications into architectures ($\text{Recall} \cap \text{Design}$), performing napkin math from memory ($\text{Recall} \cap \text{Quantify}$), evaluating architectural alternatives ($\text{Analyze} \cap \text{Design}$), sizing concrete systems ($\text{Design} \cap \text{Quantify}$), and optimizing bottlenecks ($\text{Analyze} \cap \text{Quantify}$). In *interview* construction, compound zones and mastery carry the strongest Staff+ signal; pure zones and sparse realization are still essential for warm-up, screening, and corpus organization, as we justify in Section 3.

wall. What remains defensible is mechanical sympathy, that is, quantitative reasoning grounded in real hardware specifications. We use *physics-grounded* throughout to mean that every question has a numerical answer derivable from such specifications, drawing its constants from the companion MLSysIM framework (Reddi, 2026c). Yet interview-preparation resources for this role are scarce. LeetCode (LeetCode Inc., 2024) alone lists 2,800+ algorithmic coding problems, yet the mainstream prep stack still under-teaches GPU memory hierarchies, KV-cache sizing, and deployment-scale physics. Section 12 compares that landscape to STAFFML in detail. Huyen’s *Designing Machine Learning Systems* (Huyen, 2022) provides strong conceptual coverage but no quantitative practice. Mock interview platforms focus on behavioral and coding rounds, not on system design with *napkin math*, the rapid back-of-the-envelope calculations that ML systems engineers use to estimate performance, capacity, or cost from first principles in real time. The duty-cycling box below shows the canonical shape; throughout this paper, every napkin-math block follows the same pattern of pulling a formula

and its constants from memory, plugging in numbers, and reading off a magnitude. The result: candidates study the wrong material, and interviewers lack a validated question bank calibrated to the cognitive demands of systems reasoning.

STAFFML rests on one thesis: **constraints drive architecture**. You do not choose a Transformer because it is trendy; you choose it because its attention mechanism parallelizes across thousands of cores on real silicon, unlike the sequential dependency chains of RNNs. You do not use mixed precision because a tutorial told you to; you use it because FP16 tensor cores deliver $2\times$ the throughput of FP32 and halve the memory footprint, which can change whether your model fits on one GPU or requires four. Concretely, here is what a STAFFML question looks like, lifted from the TinyML track:

duty-cycling
fluency • tinyml • L3

Scenario: Your wildlife camera classifies birds on a Cortex-M4, waking once per second for a 20 ms inference at 50 mW

active power. Sleep power is 10 μ W. Powered by two AA batteries (3,000 mAh at 3 V), how long will the device last?

Napkin math: Average power at 2% duty cycle:

$$\underbrace{0.02}_{\text{duty}} \times \underbrace{50}_{\text{mW}} + \underbrace{0.98}_{\text{sleep}} \times \underbrace{0.01}_{\text{mW}} \approx 1.01 \text{ mW}$$

Battery: 9,000 mWh $\div$ 1.01 mW $\approx$ 8,900 hours $\approx$ 1 year.

Answer: Two AA batteries sustain a year of continuous monitoring. At this duty cycle the active term (1.0 mW) dominates the sleep term (0.01 mW); halving the duty cycle to 1% nearly doubles the lifetime, while driving sleep current to zero buys at most 1%. The constraint is duty cycle, not silicon compute. Section 10 presents derivations across four cognitive zones.

Every STAFFML question has this shape: a paragraph-sized scenario, a common mistake the candidate is likely to make, a napkin-math derivation grounded in real hardware specs, a one-sentence answer, and a four-axis classification (topic, cognitive mode, deployment regime, level) printed as the metadata bar. A subset of questions, the *visual* items, additionally pair the scenario with a diagram (a roofline plot, a pipeline-bubble timeline, a KV-cache page layout, or an interconnect topology) when the reasoning act depends on reading a picture rather than parsing prose; Section 10 works one such item end-to-end. This philosophy, drawn from the *Machine Learning Systems* textbook (Reddi, 2026b), insists that AI is infrastructure, and infrastructure has laws. Five laws govern ML systems architecture:

1. **The Roofline Bound** (Williams et al., 2009). Every workload is either compute-bound or memory-bandwidth-bound, determined by its arithmetic intensity relative to the hardware’s ridge point.
2. **The Memory Capacity Wall.** Model weights, optimizer state, activations, and KV-cache must fit simultaneously in available memory; exceeding capacity forces parallelism, offloading, or compression.
3. **The Communication Tax.** In distributed systems, gradient synchronization, tensor transfers, and pipeline bubbles consume time proportional to model size and inversely proportional to interconnect bandwidth.
4. **The Power Envelope.** Sustained throughput is bounded by thermal dissipation, not peak compute; real performance is the minimum of compute capability and cooling capacity.

5. **The Cost Multiplier.** Every architecture decision compounds through fleet size, electricity cost, and amortization period; a $2\times$ efficiency improvement at 1,000 GPUs saves millions of dollars annually.

These laws are the paper’s organizing pillars, not a fifth axis: the topic taxonomy is shaped by which laws each concept exercises, but the corpus does not separately tag a question with the laws it touches. Operationalizing the laws as a queryable axis is left to future work. STAFFML is the companion assessment tool for the *Machine Learning Systems* textbook (Reddi, 2026b), developed alongside the Harvard CS249r course (Janapa Reddi, 2024). The cognitive-mode axis is organized by the four-skill competency model summarized in Figure 1: four skills (recall, analyze, design, quantify) overlap into six compound zones and a central mastery region.

Three ways to use STAFFML. STAFFML serves three audiences:

1. **Interview preparation.** Candidates filter questions by target role level (L4–L6+), practice zone, and deployment track. Depth chains provide structured study paths from fundamentals to synthesis (Section 3.5).
2. **Interview design.** Hiring managers select questions across zones to construct balanced interview loops. The zone model provides a shared vocabulary for debriefing (for example, “strong in diagnosis, weak in specification”), which is described further in Section 11.
3. **Curriculum alignment.** Educators map course content to the topic taxonomy to see which competency cells their curriculum covers and where gaps remain; the prerequisite graph in Section 4 suggests teaching order.

Figure 2 previews what those three audiences actually see: a single practice session in the STAFFML web interface. The filter rail constrains the pool (axis-aligned filters plus session toggles such as chains-only and napkin-math-only); the main column shows the question card, including chain position and, after a reveal, the model answer and self-rating; and the tools rail offers references and the Ask Interviewer clarification panel.

The remainder of the paper traces the derivation chain from competency claim to corpus. Section 2 introduces the backward-design methodology that derives the assessment space before any items are

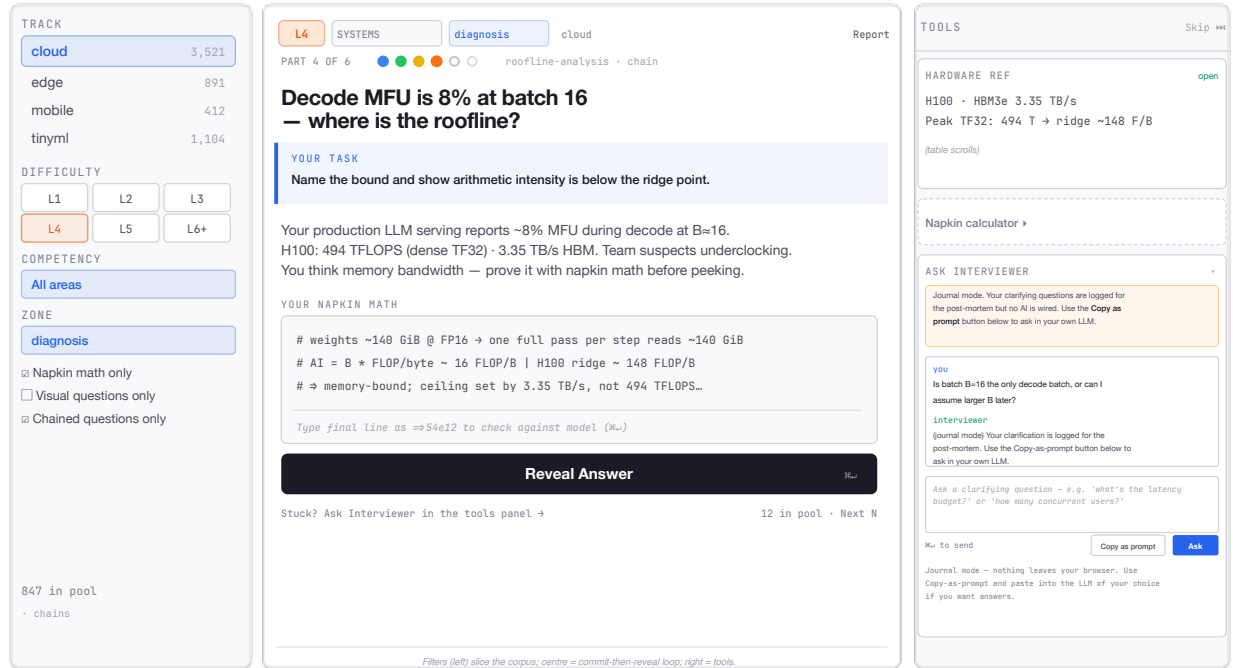


Figure 2. A practice session in the STAFFML interface. Filter sidebar (left) drives the corpus slice the center card draws from, including the stored axes (track, level, competency, zone) and session toggles (for example, chains-only, napkin-math-only, and visual questions) alongside them. The main column (center) shows the diagnosis-zone L4 roofline-analysis item (see Section 10): chain position, *Your task*, scenario, and napkin-math input in a pre-reveal frame, followed by *Reveal Answer*; after reveal, the UI surfaces the common mistake, model answer, and self-rating controls. The tools rail (right) provides hardware reference, the napkin calculator, and *Ask Interviewer* for Socratic follow-ups. Pool counts and the footer in the main column are session-local, consistent with our storage model.

written. Sections 3 to 6 then develop the four classification axes one at a time: the four-skill competency model and its eleven zones, the 87-topic taxonomy and prerequisite graph, the four deployment tracks and their physics regimes, and the six mastery levels mapped to Bloom’s taxonomy and industry engineering ladders. Sections 7 and 8 describe how individual questions are built against those axes and how the corpus is kept consistent: gap-driven item construction with hardware grounding and LLM-assisted drafting, then the quality pipeline (a LinkML schema as single source of truth, validate-at-write Pydantic constraints, three-stage deduplication, multi-model math verification, and a 26-check invariant suite). Section 9 reports how the published corpus is distributed across those axes, Section 10 walks through worked example questions, and Section 11 maps the model onto interview-loop design and the practice interface. Section 12 situates STAFFML against prior interview, benchmark, and educational-measurement work, and Section 13 presents the limitations and future work that remain.

2 Methodology: Backward Design

The structure of STAFFML is a derivation, not an organizational convenience. We follow the *backward design* methodology of Wiggins and McTighe (Wiggins and McTighe, 2005), which proceeds in three stages: identify the desired results, determine acceptable evidence, and only then design the assessment. The corpus size is an output of this process, not an input. We defined the competency space, filtered it by physics, bounded its capacity empirically, and let the total fall out. Figure 3 illustrates the five-step derivation chain. Each step constrains the next, and the final corpus is fully determined by the intersection of those constraints.

2.1 The Derivation Chain

Step 1. Decompose competency into skills and zones. We start from the desired result: what must a Staff-level ML systems engineer demonstrate? Mechanical sympathy is the ability to connect hardware constraints to

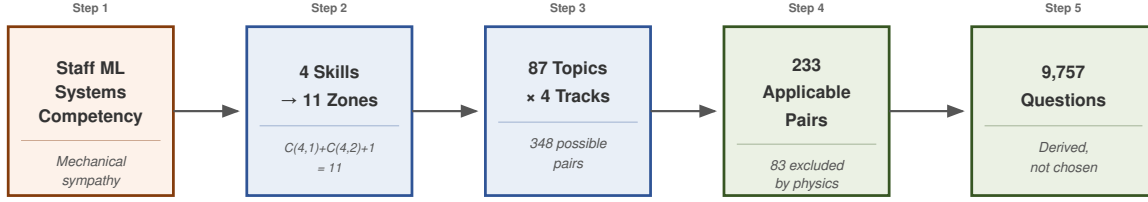


Figure 3. The backward design derivation chain. Corpus size is an output of principled constraints, not an input. Each step narrows the assessment space: competency decomposition yields 11 zones, topic curation yields 87 concepts, the physics filter excludes impossible hardware pairings, and capacity analysis bounds meaningful question density.

architectural decisions through quantitative reasoning. We decompose this into four fundamental skills (recall, analyze, design, quantify) and observe that real engineering tasks never exercise one skill in isolation. A diagnosis requires recall and analysis; napkin math requires recall and quantification. The pairwise combinations of these four skills yield six compound zones, plus one central mastery zone that exercises all four skills simultaneously, the four-circle overlap shown in Figure 1. Together these produce the 11 cognitive zones described in Section 3.

The zone model complements Bloom’s revised taxonomy (Anderson et al., 2001). Bloom’s two-dimensional framework (cognitive process $\times$ knowledge type) captures the depth of cognitive engagement but not the simultaneous exercise of multiple cognitive modes that characterizes real engineering tasks. Consider fluency (recall $\cap$ quantify): an engineer who retrieves hardware specifications from memory and executes napkin math in real time. Bloom has no single level for this act. The four-skill model treats skills as dimensions, not levels. The zones are the cells of a combinatorial lattice: $\binom{4}{1} + \binom{4}{2} + \binom{4}{4} = 4 + 6 + 1 = 11$.

Step 2. Define topic coverage. We identify 87 topics spanning 13 competency areas as the minimum spanning set of Staff-level ML systems knowledge. The prerequisite graph (57 edges) encodes dependencies.

Step 3. Apply the deployment-feasibility filter. Not every topic has a physical substrate on every hardware tier. We derive an *applicability matrix* that excludes 83 topic-track pairs with one-sentence justifications grounded in the

relevant physics and deployment constraints (e.g., a 256 KB MCU has no place for an RDMA-fabric question; a datacenter inference cluster has no place for an MCU duty-cycling question). The remaining 233 pairs define the valid assessment space. This establishes *content validity* (Messick, 1995): every question references a concept with physical meaning on the targeted hardware tier.

Step 4. Bound capacity empirically. For each valid cell, how many meaningfully distinct questions exist? Capacity varies by cognitive complexity and difficulty level. We treat this as an empirical question: the published corpus and coverage statistics (Section 9) make density visible without fixing a prior quota per cell.

Step 5. Lock structure before scaling. The taxonomy, applicability matrix, and competency tags are treated as part of the assessment specification: they are versioned with the schema so every item maps to a defensible coordinate in the assessment space.

Content validity is structurally enforced by the pipeline. Construct validity, whether zones capture psychometrically distinct competencies, awaits candidate response data and is discussed in Section 13. The next section details the derivation chain’s foundational step: the four-skill competency model.

3 The StaffML Competency Model

Step 1 of the derivation chain held that Staff-level ML systems expertise decomposes into four skills whose intersections produce the cognitive zones the assessment is built on. This section is that decomposition. We first motivate why a single-skill model is insufficient, then introduce the four fundamental skills,

derive the eleven cognitive zones their pairwise intersections produce, describe the affinity between zones and Bloom’s mastery levels, and finally show how zone composes with the other three classification axes (topic, track, and level) into the operational four-axis framework that organizes the rest of the paper.

To motivate the decomposition, consider that a software engineer who can implement a Transformer from scratch (Reddi, 2026d) may still be unable to answer a deceptively simple question: “How much VRAM does that model need, and why does that number determine your entire serving architecture?” The gap is not knowledge of algorithms; it is mechanical sympathy, the ability to reason quantitatively about how software interacts with physical hardware. An engineer who recalls every hardware specification but cannot analyze a failure is incomplete; one who designs elegant architectures but cannot produce napkin math to validate them is dangerous. The competency model must capture the intersection of these modes, not any one of them in isolation. The TinyTorch competency matrix (Reddi, 2026d) defines what to teach through 8 knowledge areas crossed with 5 capability levels; the STAFFML four-skill model defines how to assess whether those competencies have been acquired.

3.1 Four Fundamental Skills

We decompose ML systems expertise into four skills whose intersections define qualitatively different cognitive acts. The four-circle Venn structure used to visualize these intersections (Figure 1) is borrowed from the *ikigai* life-philosophy diagram (the overlap of passion, skill, need, and reward); only the geometric form is taken, the content is our own.

1. **Recall.** Facts, definitions, hardware specifications. “What is the HBM bandwidth of an H100?” The engineer retrieves a stored constant.
2. **Analyze.** Tradeoffs, reasoning, root cause analysis. “Why is this workload memory-bound?” The engineer reasons about the interaction between compute and data movement.
3. **Design.** Architecture decisions, requirements-to-system mapping. “Design a serving stack with $P99 < 100\text{ms}$.” The engineer translates constraints into an architecture.
4. **Quantify.** Napkin math & estimation. “Calculate the VRAM required for a 70B model at

128K context.” The engineer produces a specific number from a formula and hardware specs.¹

Figure 1 renders this list geometrically: each labeled overlap is a compound assessment mode (two skills at once), and the center is mastery (all four).

These four skills are not Bloom’s levels; they are orthogonal dimensions of expertise. A senior engineer may have deep recall and strong quantification skills (fluency) but weak design ability; another may excel at analysis and design (evaluation) but struggle with napkin math. The four-skill model makes these profiles visible and assessable.

3.2 Eleven Cognitive Zones

The four skills combine into 11 zones: 4 pure zones (each exercising a single skill), 6 compound zones (each exercising two skills simultaneously), and 1 mastery zone (all four). Table 1 lists each zone with its constituent skills, the Bloom-level range it typically lands in, and a one-line description.

The eleven zones are not all of equal interview value. The five compound zones (diagnosis, specification, fluency, evaluation, optimization), together with mastery, carry the strongest signal for Staff+ assessment, because these compound acts are precisely what distinguish a candidate who reasons across cognitive modes from one who can execute each mode in isolation. The four pure zones (recall, analyze, design, quantify) function primarily as corpus organization categories and warm-up question pools; realization (design $\cap$ quantify), while well-defined, occurs sparsely because most sizing questions also require recall and therefore land in fluency or specification (Section 11 returns to how this asymmetry shapes interview-loop construction). We retain all eleven zones in the schema so the corpus can be sliced by cognitive mode for organization and authoring, but readers focused on assessment design can treat the six interview-relevant zones as the operative set.

We deliberately omit the four possible three-skill intersections (e.g., recall $\cap$ analyze $\cap$ design) because, in practice, questions exercising three skills almost always exercise all four, making them mastery questions. The two-skill compound zones represent the cognitively distinctive acts; adding three-skill zones would dilute rather than sharpen the classification.

Pure zones test a single skill in isolation:

¹The schema identifier for this zone is `implement`; we use *quantify* in prose because the act being assessed is producing a number, not writing production code.

Table 1. Zone definitions, typical level bands, and the cognitive act each zone tests. The 11 cognitive zones decompose into four pure zones (single skill), six compound zones (two skills exercised simultaneously), and one mastery zone (all four). The *Typ. levels* column is a soft authoring prior rather than a hard validation rule; the hard rule is the zone–Bloom matrix in Section 8, which forbids cognitively incoherent pairings such as a recall question tagged at the evaluate level.

Zone	Skills	Typ. levels	Cognitive act being tested
Recall	recall	L1–L2	Retrieve facts, definitions, and hardware specifications without analysis or computation. The skill of remembering specs cold.
Analyze	analyze	L3–L4	Reason qualitatively about system behavior and weigh tradeoffs without committing to specific numbers.
Design	design	L4–L5	Map requirements onto an architecture without yet committing to specific quantitative choices.
Quantify	quantify	L2–L3	Apply a given formula to given constants and produce a specific number, with no architectural judgment required.
Diagnosis	recall + analyze	L3–L4	Identify a root cause by recalling hardware specifications and reasoning about which bottleneck explains the observed symptom.
Specification	recall + design	L4–L5	Translate a quantitative constraint (a latency SLO, a memory budget, a throughput target) into a concrete architectural choice.
Fluency	recall + quantify	L2–L3	Recall a formula and the relevant constants, then execute the arithmetic in real time without consulting references. The signature phone-screen skill.
Evaluation	analyze + design	L5–L6+	Compare alternative architectures against fixed constraints and justify the choice with quantitative reasoning.
Realization	design + quantify	L5–L6+	Design a configuration and produce the numbers that make the configuration feasible on the available hardware.
Optimization	analyze + quantify	L4–L5	Locate a bottleneck and quantify the impact of a specific remedy such as kernel fusion, quantization, or offloading.
Mastery	all four	L6+	Full Staff-level synthesis. One problem requires diagnosing a failure, designing a fix, sizing the resources, and estimating the cost.

- **Recall.** “What is the ridge point of an H100?” Retrieve a fact.
- **Analyze.** “Why does GPU utilization collapse at batch size 1?” Reason about system behavior.
- **Design.** “Architect a KV-cache paging system.” Produce an architecture.
- **Quantify.** “Write the formula for KV-cache memory.” Produce a concrete artifact.

Compound zones test two skills simultaneously. These are the zones that most sharply differentiate strong from weak candidates, because they require integrating knowledge across cognitive modes:

- **Diagnosis** (Recall $\cap$ Analyze). “MFU is 40%, diagnose it.” The candidate must recall hardware specifications and reason about which bottleneck explains the gap. This is the “3 AM on-call” skill.
- **Specification** (Recall $\cap$ Design). “P99 < 100 ms with 70B model, design it.” The candidate must recall hardware capabilities and translate the latency constraint into an architecture (tensor parallelism degree, batch size, KV-cache budget).
- **Fluency** (Recall $\cap$ Quantify). “How much VRAM does a 70B model need?” The candidate must recall the formula and hardware specs,

then execute the arithmetic. This is napkin math from memory, the signature skill of Staff+ engineers.

- **Evaluation** (Analyze $\cap$ Design). “MoE vs. dense: which architecture for this serving workload?” The candidate must analyze the tradeoffs and reason about which design better satisfies the constraints. No recall is required; the specs are given.
- **Realization** (Design $\cap$ Quantify). “Size the micro-batches for this pipeline-parallel training job.” The candidate must design the pipeline schedule and produce concrete numbers (batch splits, memory per stage, bubble fraction).
- **Optimization** (Analyze $\cap$ Quantify). “Fix the memory wall.” The candidate must analyze the bottleneck and produce a concrete fix (kernel fusion, quantization, offloading) with measurable improvement.

Mastery (all four skills) represents full Staff-level synthesis: “Your LLM serving cluster’s TTFT spiked from 100 ms to 4 s after a traffic surge. Diagnose the root cause, design a fix, size the required resources, and estimate the cost.” This zone requires

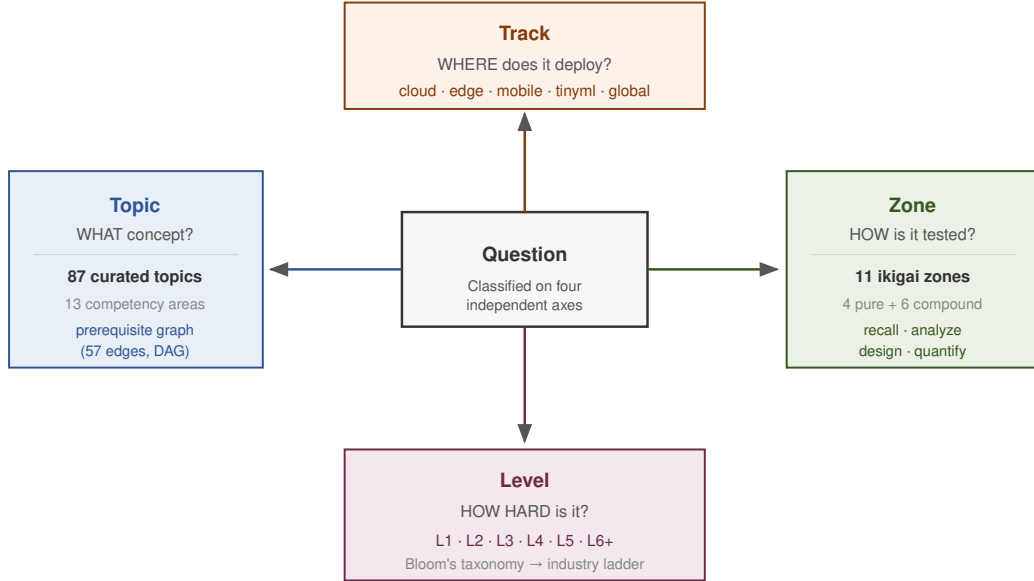


Figure 4. Four-axis classification. Every question is independently classified by *topic* (what concept is tested), *zone* (how the concept is tested), *track* (which physics regime applies), and *level* (what cognitive depth is required). The axes are orthogonal: any valid combination defines a unique assessment cell.

recalling hardware specs, analyzing the failure mode, designing an architectural solution, and executing the napkin math to validate it.

3.3 Zone–Level Affinity

Each zone has a natural affinity with certain mastery levels, though the mapping is not rigid. The *Typ. levels* column of Table 1 summarizes the affinity: this is a soft authoring prior, not a hard validation rule, and is not a fifth axis layered on top of the four-skill model. The diagram that places level alongside topic, zone, and track appears in the next subsection (Figure 4), where we define the operational four-axis framework.

The hard rule that actually rejects items at write time is the *zone–Bloom matrix*, not the level affinity column. The level column is descriptive: it indicates that a fluency question typically reads as L2–L3, but no commit is rejected for tagging a fluency item as L4. By contrast, an evaluation zone paired with a Bloom level of *Remember* or *Understand* is structurally incoherent (the cognitive act of evaluation requires at least *Apply*-level engagement), and the Pydantic validator rejects such combinations at commit time. Section 8 describes this hard rule and the broader invariant suite.

3.4 The Four-Axis Classification

The competency model answers the question of which cognitive skills matter. To operationalize this as a practical assessment framework, STAFFML classifies each question along four independent axes, each capturing a distinct dimension of ML systems expertise. Figure 4 summarizes the four stored coordinates (topic, zone, track, and level), including how level sits beside the other three.

The **topic axis** identifies which concept a question tests. The 87 topics are organized into 13 competency areas (the area-level rollup is in Table 2, with the full topic-by-area listing in Table 3), from memory hierarchies and parallelism strategies to power budgeting and reliability, and connected by 57 typed edges (prerequisite, broader, narrower, related) that encode the dependency structure of ML systems knowledge.

The **zone axis** captures how a concept is tested, using the 11 cognitive zones defined above. It determines the cognitive profile of a question independently of its subject matter.

The **track axis** specifies where the scenario takes place, which physics regime governs the problem. Cloud questions operate at PFLOPS scale under memory bandwidth and network fabric constraints. Edge questions operate under thermal envelopes and real-time deadlines. Mobile questions face battery budgets and shared-resource contention. TinyML

questions confront SRAM limits and hard real-time on bare metal. A fifth “global” track captures cross-cutting principles.

The **level axis** encodes cognitive depth, mapped to Bloom’s taxonomy (Anderson et al., 2001) (L1 Remember through L6+ Create) and aligned with engineering ladders from intern through Staff+.

These four axes are orthogonal by design. A question about KV-cache management (topic) can test fluency (zone) in the cloud regime (track) at L3 difficulty (level), or it can test diagnosis (zone) in the edge regime (track) at L4 difficulty (level). The three-dimensional coverage cube of topic $\times$ zone $\times$ track contains $87 \times 11 \times 4 = 3,828$ cells (excluding the cross-cutting global track). While level acts as an independent coordinate, it exhibits a strong natural affinity for the zone axis (recall: questions skew L1–L2; mastery skews L6+), which we enforce as an authoring prior. Section 9 carves the cube down to 2,563 meaningful cells via the applicability matrix; the combinatorial structure enables the precise gap analysis we apply there.

3.5 Depth Chains

Questions on the same topic are linked into *depth chains* that progress through Bloom’s levels, testing increasingly sophisticated cognitive acts. A chain ensures that assessment covers the full range from recall to synthesis. The corpus contains 843 chains spanning multiple levels.

Each step in the chain builds on the previous, and the zone shifts from recall through quantification to diagnosis and finally mastery. Chains serve three purposes: (1) validating that the topic taxonomy supports questions at multiple cognitive levels, (2) enabling progressive study paths where candidates build from fundamentals to synthesis, and (3) providing interviewers with calibrated question sequences that reveal the candidate’s ceiling.

For example, a chain on KV-cache management progresses through six levels and six zones:

- L1 (Recall):** “What is a KV-cache and why does it grow with sequence length?”
- L2 (Quantify):** “Calculate the KV-cache size for Llama-70B at 128K context.”
- L3 (Fluency):** “Your 80 GB GPU OOMs at 128K context. Derive the memory breakdown.”
- L4 (Diagnosis):** “PagedAttention reduces memory by 60%. Explain the mechanism.”
- L5 (Specification):** “Design a KV-cache compression strategy. What precision trade-offs are acceptable?”
- L6+ (Mastery):** “Your paged KV-cache suffers frag-

Table 2. Competency areas. The 13 areas containing 87 curated topics, organized by systems domain. The specific topics that populate each area are listed in Table 3.

Area	Scope	Topics
Architecture	System design patterns	8
Compute	Accelerators, execution models	7
Cross-cutting	Compound AI, MLOps, responsible AI, sustainability	8
Data	Pipelines, storage, preprocessing	7
Deployment	Serving, compression, adaptation	9
Latency	Scheduling, prefill/decode, SLOs	6
Memory	VRAM, KV-cache, offloading	8
Networking	Collective ops, interconnects, fabric	6
Optimization	Kernel fusion, graph rewrites, profiling	7
Parallelism	Data/tensor/pipeline/3D parallelism	7
Power	Thermal, energy, battery, DVFS	5
Precision	Quantization, mixed precision, formats	3
Reliability	Fault tolerance, checkpointing, recovery	6
Total		87

mentation under variable-length batching. Diagnose, design a fix, and size the memory overhead.”

The corpus contains 843 topic-level chains. Counting by the number of linked questions stored in the chain registry, 147 have two, 355 have three, 234 have four, 82 have five, and 24 have six; 106 have five or more linked questions (the same “full” count emitted by the release exporter). In total, 2,849 questions (29.9% of the corpus) participate in at least one chain.

4 Topic Taxonomy

The competency model fixes *how* a question exercises the candidate; the topic axis fixes *what* concept the question is about. The two are independent by design, which is why we treat the topic taxonomy as its own section: every zone needs to be populated by topics that admit Staff-level reasoning, and every topic needs to be reachable through more than one zone. This section describes the curated topics, their prerequisite relationships, and the knowledge organization methodology that governs selection and maintenance.

4.1 Curated Topics

The STAFFML taxonomy contains 87 topics organized into 13 competency areas (Table 2). Topics must satisfy three inclusion filters:

1. **Systems relevance.** Does the concept have measurable systems implications (compute, memory,

latency, throughput, cost, or power)? If you cannot write a napkin-math question about it, it does not belong.

2. **Interview currency.** Would a hiring committee at a top-5 AI lab expect a Staff ML systems engineer to reason about this concept within the next two years?
3. **Reasoning depth.** Can questions be asked at 3+ Bloom’s levels? A concept that supports only L1 recall is too shallow.

Table 2 reports the area-level rollup; Table 3 then expands each area into its constituent topics, so readers can see which specific concepts populate, say, Memory or Deployment.

4.2 The Prerequisite Graph

Topics are connected by typed edges following the SKOS vocabulary (W3C, 2009). The total edge count across the three relation types is 131, partitioned as follows:

- **Prerequisite** (57 edges). Topic A must be understood before Topic B. Example: `gpu-compute-architecture` requires `roofline-analysis` as a prerequisite, because diagnosing whether a GPU kernel is compute-bound or memory-bound requires the roofline model. The graph is verified to be acyclic with a maximum prerequisite depth of 3.
- **Broader/Narrower** (14 edges). Hierarchical containment. Example: `parallelism-strategies` is broader than `tensor-parallelism`.
- **Related** (56 edges). Lateral connection. Example: `roofline-analysis` is related to `profiling-bottleneck-analysis`.

Throughout the paper, “edges” refers to prerequisite edges by default (57 of them); broader/narrower (14) and related (56) edges are reported explicitly when relevant.

The prerequisite edges enable automated learning path generation. Given a target topic, the system computes the transitive closure of prerequisites and produces an ordered study sequence. The graph has 35 root topics (concepts with no prerequisites) and a maximum prerequisite depth of 4, ensuring that no learning path requires more than three preparatory topics. Figure 5 illustrates a subset of this graph for the parallelism area. The diamond shape is informative: data parallelism is the entry point (be-

cause every distributed-training strategy starts by replicating optimizer state across devices), tensor and pipeline parallelism extend that base in two orthogonal directions, and 3D parallelism collapses both back together. The dashed cross-area edge into gradient synchronization is the more important detail for study planning: knowledge transitively reaches across competency areas, so a candidate cannot study one column of the taxonomy in isolation and expect to be prepared for the others.

To show how the prerequisite graph guides study, consider a candidate targeting distributed training expertise. The graph suggests the following path through five topics spanning three competency areas:

```
roofline-analysis → gpu-compute-arch. →  
data-parallelism → collective-comm. →  
3d-parallelism
```

At each node, the candidate starts with L1–L2 recall and fluency questions to build foundational knowledge, then progresses to L3–L4 diagnosis and evaluation questions before moving to the next topic. The full path from roofline fundamentals to 3D parallelism mastery spans approximately 60 questions across 20 depth chains.

The prerequisite graph currently caps depth at 3. Real ML systems knowledge can require longer chains. For example, understanding why a specific collective communication pattern matters requires understanding network topology, bandwidth-latency tradeoffs, and the roofline model, a chain of depth 4 or more. Deepening the prerequisite graph is planned for future taxonomy revisions, as more topics cross the inclusion threshold and longer chains become expressible.

4.3 Knowledge Organization

The taxonomy follows the principles of faceted classification (Hjørland, 2013), where each question is independently classified on orthogonal axes rather than slotted into a single hierarchy. This design avoids the cross-classification problems inherent in monohierarchical schemes. A question about KV-cache memory pressure during continuous batching touches memory, latency, and deployment simultaneously, and a faceted system accommodates this naturally.

Topic selection follows user warrant (Soergel, 1985) rather than literary warrant. Topics are included because engineers need them during interviews, not because they appear frequently in the research literature. This distinction explains why the taxonomy

Table 3. Full topic taxonomy. All 87 curated topics organized by competency area.

Area	Topic Name	Area	Topic Name	Area	Topic Name
Architecture (8)		Data (7)		Networking (6)	
Architecture	Attention Scaling & Variants	Data	Data Efficiency & Selection	Networking	Collective Communication
Architecture	CNN Efficient Design	Data	Data Pipeline Engineering	Networking	Congestion & Flow Control
Architecture	Encoder-Decoder Tradeoffs	Data	Data Quality & Validation	Networking	Interconnect Topology
Architecture	Mixture of Experts	Data	Dataset Curation & Labeling	Networking	Load Balancing & Routing
Architecture	Model Size Estimation	Data	Feature Store & Feature Engineering	Networking	Network Bandwidth Bottlenecks
Architecture	Neural Architecture Search	Data	Storage Format & Selection	Networking	RDMA & High-Performance Transport
Architecture	Recommendation Systems Eng.	Data	Streaming & Real-Time Ingestion	Optimization (7)	
Architecture	Transformer Systems Cost	Deployment (9)		Optimization	Graph Compilation & Optimization
Compute (7)		Deployment	A/B & Rollout Strategies	Optimization	IO-Aware Attention
Compute	Accelerator Comparison	Deployment	Compound AI Systems	Optimization	Kernel & Operator Fusion
Compute	Chiplet Architecture	Deployment	Container & Cluster Orchestration	Optimization	Knowledge Distillation
Compute	Compute Cost Estimation	Deployment	Disaggregated Serving	Optimization	Operator Scheduling
Compute	GPU Compute Architecture	Deployment	MLOps Lifecycle	Optimization	Pruning & Sparsity
Compute	MCU Compute Constraints	Deployment	Model Adaptation Systems	Optimization	Speculative Decoding
Compute	Roofline Analysis	Deployment	Model Format & Runtime Conversion	Parallelism (7)	
Compute	Systolic Arrays & Dataflow	Deployment	Model Serving Infrastructure	Parallelism	3D Parallelism
Cross-Cutting (8)		Deployment	OTA & Firmware Updates	Parallelism	Comm./Compute Overlap
Cross-Cutting	Autograd & Comp. Graphs	Latency (6)		Parallelism	Data Parallelism
Cross-Cutting	Differential Privacy	Latency	Batching Strategies	Parallelism	Gradient Synchronization
Cross-Cutting	Fairness & Bias Evaluation	Latency	Latency Decomposition	Parallelism	Model & Tensor Parallelism
Cross-Cutting	Federated Learning	Latency	Profiling & Bottleneck Analysis	Parallelism	Pipeline Parallelism
Cross-Cutting	Responsible AI & Governance	Latency	Queueing Theory	Parallelism	Scheduling & Resource Management
Cross-Cutting	Software Portability	Latency	Real-Time Deadlines	Precision (3)	
Cross-Cutting	Sustainability & Carbon Acct.	Latency	Tail Latency & SLAs	Precision	Extreme Quantization
Cross-Cutting	TCO & Cost Modeling	Power (5)		Precision	Mixed-Precision Training & Inference
Memory (8)		Power	Datacenter Efficiency	Precision	Quantization Fundamentals
Memory	Activation Memory & Checkpointing	Power	Duty Cycling & Energy Harvesting	Reliability (6)	
Memory	DMA & Data Movement	Power	Energy Per Operation	Reliability	Adversarial Robustness & Security
Memory	KV-Cache Management	Power	Power Budgeting	Reliability	Distribution & Drift Detection
Memory	Memory Hierarchy Design	Power	Thermal Management	Reliability	Fault Tolerance & Checkpointing
Memory	Memory Pressure Management			Reliability	Graceful Degradation
Memory	Memory-Mapped Inference			Reliability	Monitoring & Observability
Memory	Tensor Arena Planning			Reliability	Safety & Certification
Memory	VRAM Budgeting				

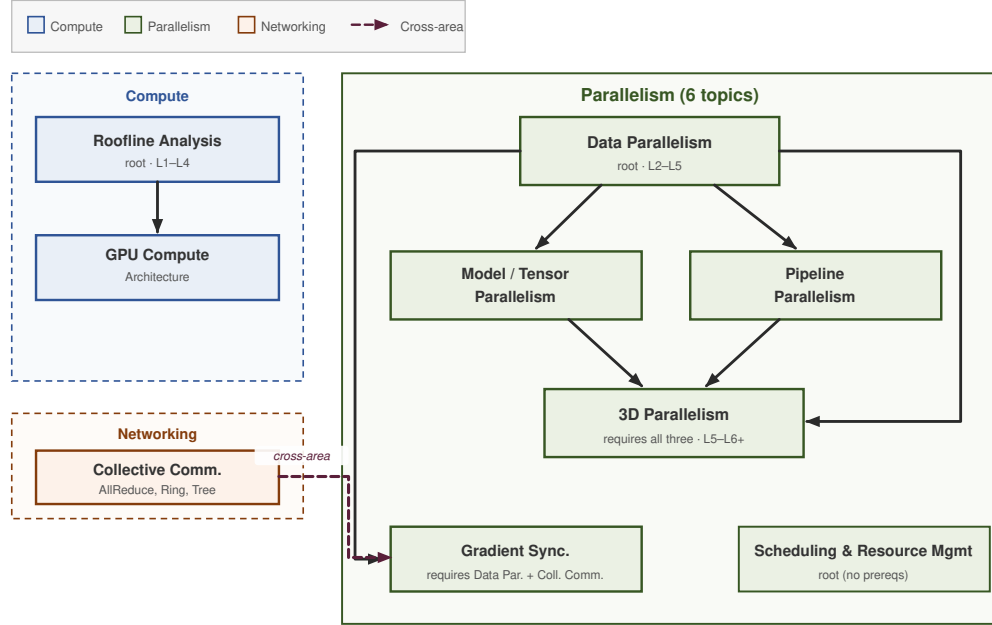


Figure 5. Prerequisite subgraph: parallelism area. Six of 87 topics showing the dependency diamond: Data Parallelism fans out to Model/Tensor Parallelism and Pipeline Parallelism, which converge on 3D Parallelism. The dashed cross-area edge from Collective Communication (networking) to Gradient Synchronization illustrates how the prerequisite graph connects competency areas. The full graph has 57 edges, maximum depth 4, and is verified acyclic.

includes `tco-cost-modeling` (rarely a research topic, always an interview topic) but excludes attention mechanism variants (heavily researched, rarely asked as systems questions).

The taxonomy is maintained against signals from industry practice and systems research (e.g., job postings, MLSys-class venues). New topics enter only if they satisfy the three inclusion filters and support questions at multiple mastery levels.

4.4 Example Topic with Edges

To illustrate the structure, consider the topic `gpu-compute-architecture`:

ID: `gpu-compute-architecture`
Area: Compute
Description: GPU execution model: warps, thread blocks, occupancy, Tensor Cores, and memory coalescing.
Tracks: cloud, edge
Edges:
 requires → `roofline-analysis`
 related → `kernel-fusion`
 related → `systolic-dataflow`

The prerequisite edge encodes that understanding the roofline model is necessary before reasoning about GPU compute architecture, because determin-

ing whether a kernel is compute-bound or memory-bound requires roofline analysis. The edges capture lateral connections: kernel fusion decisions depend on GPU architecture, and systolic arrays are an alternative accelerator execution model worth comparing.

5 Deployment Tracks

The taxonomy says *what* a question tests. The track axis indicates *where* the scenario takes place, because a concept does not behave the same way on every hardware tier. KV-cache management on an H100 is bounded by HBM bandwidth at TB-per-second scale; the same concept on a Cortex-M4 is bounded by 256 KB of on-chip SRAM. Without an explicit deployment axis, the taxonomy would erase that gap and let questions drift into hardware regimes where their physics no longer applies.

Why not simply divide the space into “training” and “inference”? Because tracks are not workload labels; they are different **physics envelopes** that govern the napkin math. Cloud workloads are bandwidth- and network-bound; edge workloads are thermal-bound; mobile workloads are battery-bound; and TinyML workloads are strictly SRAM-bound. A can-

Table 4. Deployment tracks and physics regimes. Each track maps to distinct dominant constraints, characteristic scales, and failure modes that govern the napkin math.

Track	Dominant Constraint	Scale	Unit	Failure Mode	Napkin Math Involves
Cloud	Memory BW + network fabric	PFLOPS	TB/s	Throughput collapse	TFLOPS, TB/s, \$/GPU-hr
Edge	Thermal envelope + real-time	TOPS	Watts	Deadline miss	TOPS, Watts, milliseconds
Mobile	Battery + shared resources	TOPS/W	mAh	Battery drain	TOPS/W, mAh, background limits
TinyML	SRAM + hard real-time	MFLOPS	KB	Stack overflow	MFLOPS, KB, microseconds
Global	Cross-cutting principles	–	–	–	Depends on context

didate who understands KV-cache sizing in the cloud cannot trivially transfer that knowledge to an SRAM-constrained microcontroller; the physical constraints dictate entirely different engineering tradeoffs.

Table 4 summarizes the five deployment tracks. Four (cloud, edge, mobile, TinyML) correspond to distinct hardware regimes; the fifth (“global”) captures cross-cutting principles. The same concept (KV-cache management, say) requires different reasoning, different formulas, and different napkin math depending on the deployment target.

5.1 Cloud: Bandwidth and Fabric

In the cloud regime, compute is abundant but data movement is the bottleneck. An H100 GPU delivers 989 TFLOPS FP16 dense (1,979 with 2:4 structured sparsity) but only 3.35 TB/s of HBM bandwidth. The roofline ridge point at TF32 is $494/3.35 \approx 148$ FLOPs/Byte. Most LLM inference workloads, particularly the autoregressive decode phase, operate well below this ridge point, making them memory-bandwidth-bound despite the massive compute available. The dominant engineering challenges are KV-cache management, continuous batching (Yu et al., 2022) to maximize GPU utilization, and inter-node communication for distributed training where InfiniBand fabric at 400 Gb/s still becomes the bottleneck during AllReduce operations.

Cloud questions test reasoning about throughput, latency, and cost at datacenter scale. Napkin math involves TFLOPS, TB/s, and dollars per GPU-hour.

5.2 Edge: Thermal and Real-Time

Edge deployment (autonomous vehicles, robotics, industrial inspection) operates under a thermal envelope that caps sustained throughput. An NVIDIA Orin delivers 275 TOPS INT8 within a 60 W power budget, but sustained operation at maximum throughput may require thermal throttling. The main constraint is the intersection of compute budget and real-time deadlines. The perception pipeline

must process a camera frame within 33 ms (30 fps) regardless of scene complexity, leaving no room for the tail-latency spikes that cloud systems tolerate.

Edge questions test reasoning about worst-case execution time, power budgets, and the tradeoff between model accuracy and inference latency. Napkin math involves TOPS, Watts, and milliseconds.

5.3 Mobile: Battery and Resources

Mobile ML operates on shared silicon: the neural processing unit (NPU) shares power, thermal headroom, and memory bandwidth with the CPU, GPU, camera ISP, and cellular modem. A smartphone’s ~5,000 mAh battery must last all day, meaning any ML workload must justify its energy cost against competing demands. The dominant constraint is TOPS per Watt. A current-generation mobile NPU may deliver 30–40 TOPS at 4–5 W (e.g., Apple’s A17 Pro Neural Engine and Qualcomm’s Snapdragon 8 Gen 3 Hexagon NPU), but running it continuously alongside the CPU, camera pipeline, and memory subsystem draws closer to 10 W of system power and would drain the battery in well under 2 hours.

Mobile questions test reasoning about energy budgets, model compression for on-device deployment, and the scheduling tradeoffs of running ML inference alongside other system tasks.

5.4 TinyML: SRAM and Real-Time

TinyML represents the most constrained physics regime. A Cortex-M4 microcontroller operates at ~100 MFLOPS with 256 KB of SRAM and no operating system (Lin et al., 2020). There is no virtual memory, no swap space, and no dynamic allocation; the model, activations, and runtime must all fit in SRAM simultaneously. The dominant constraint is hard real-time. A keyword spotting model must classify an audio frame within a fixed time budget, because missing a deadline means dropping audio samples with no recovery mechanism.

TinyML questions test reasoning about model footprint in kilobytes, fixed-point arithmetic, and the systems engineering required to run inference on bare metal. Napkin math involves MFLOPS, KB, and microseconds.

5.5 Cross-Track Variation

The track structure ensures that questions test regime-appropriate reasoning. Consider KV-cache management across tracks:

- **Cloud.** “A 70B model’s KV-cache at 128K context is ~42 GB. How many H100s do you need?” (Memory capacity reasoning at TB scale)
- **Edge.** “Your ADAS model caches 10 frames of detection history. At 30 fps on Orin’s 32 GB LPDDR5, what is the memory pressure?” (Real-time memory budget reasoning)
- **TinyML.** “Your keyword spotter needs 2 KB of rolling buffer on a 256 KB MCU. What fraction of SRAM does this consume, and what is left for the model?” (SRAM partitioning at KB scale)

All three questions test the same underlying concept (caching intermediate computation to avoid re-computation) but the physics, the formulas, and the failure modes are entirely different.

6 Mastery Levels

The track axis fixes the physics regime, but a question still has to land at the right cognitive depth for the candidate it targets: an L4 senior should not be screened on L1 recall, and an intern should not be asked to architect a multi-cluster training job. The fourth axis, *level*, encodes that depth. STAFFML divides cognitive depth into 6 mastery levels, building on the dimension introduced in Section 3 and giving it both an educational anchor (Bloom’s revised taxonomy) and an industry anchor (engineering ladders).

The 6 mastery levels map to Bloom’s revised taxonomy (Anderson et al., 2001) and are explicitly designed to correspond to industry engineering ladders. This dual mapping (cognitive depth and professional seniority) makes STAFFML practical both as an educational tool and as interview preparation calibrated to specific hiring levels. Table 5 states the Bloom’s category, typical ownership scope, and representative industry ladder band for each level.

Each level tests a qualitatively different cognitive operation. L1 requires memorization of physical constants (“HBM3 latency is ~300 ns”). L3 requires

Table 5. Mastery levels. Bloom’s taxonomy mapped to industry engineering ladders. The *Industry Ladder* column is illustrative, not normative: it reflects rough leveling analogues observed at companies like Google (L3–L7) and Meta (E3–E7), but real engineering ladders span multiple cognitive demands at each band, and individual companies’ rubrics vary. The mapping is an authoring prior, not an empirically calibrated correspondence; psychometric calibration against ladder bands is acknowledged as ongoing work (Section 13).

Level	Bloom’s	Scope	Industry Ladder (illust.)
L1	Remember	Own a task	Intern / New Grad (L3)
L2	Understand	Own a task	Junior (L3–L4)
L3	Apply	Own a component	Mid-level (L4)
L4	Analyze	Own a component	Senior (L5)
L5	Evaluate	Own a system	Staff (L6)
L6+	Create	Own architecture	Senior Staff+ (L7+)

applying a formula to a novel scenario (“Calculate KV-cache size for Llama-70B at 128K context”). L6+ requires synthesizing novel architectures from first principles (“Design a scheduling policy that prevents prefill starvation under continuous batching”). At this tier the constraint relationship also becomes bidirectional: engineers not only optimize software for fixed hardware, they shape hardware procurement, cluster topology, and custom silicon (“Given this workload’s arithmetic intensity, what must the ridge point of our next-generation accelerator be?”).

The industry ladder mapping is intentional, not incidental. When a hiring manager at Google prepares an L5 (Senior) interview loop, they can filter STAFFML for L4–L5 questions and know that the cognitive demands match the leveling expectations. When a candidate preparing for a Staff (L6) role wants to practice, they can target L5–L6+ questions that require the systems-level synthesis expected at that level. This alignment between Bloom’s cognitive framework and industry practice is what makes the level axis actionable for both sides of the interview table.

We adopt Bloom’s revised taxonomy (Anderson et al., 2001) over alternatives such as Webb’s Depth of Knowledge (Webb, 1997) and the SOLO taxonomy (Biggs and Collis, 1982) for three reasons. First, Bloom’s six levels align naturally with the six tiers of industry engineering ladders, providing an intuitive mapping that interviewers and candidates already understand. Second, Bloom’s cognitive verbs (remember, understand, apply, analyze, evaluate, create) map directly to the cognitive operations demanded

by systems interview questions, making level assignment more reliable than the content-area-dependent DOK levels. Third, the revised taxonomy’s two-dimensional structure (cognitive process $\times$ knowledge type) complements rather than conflicts with the orthogonal zone axis. Bloom’s captures depth, while zones capture cognitive mode.

The level distribution in the corpus reflects intentional design decisions. L3–L4 questions (Apply and Analyze) are the most numerous because they correspond to the cognitive sweet spot of technical interviews: complex enough to differentiate candidates, but tractable within a 45-minute interview slot. L1–L2 questions serve as warm-up and screening, while L6+ questions are reserved for Staff+ interview loops where candidates are expected to synthesize novel solutions.

7 Item Construction

With the four classification axes specified, the next question is how individual items are built to occupy them. We organize item construction around four ideas, taken in turn by the subsections that follow. Evidence-Centered Design specifies what each question must measure before it is written; hardware grounding ties every scenario to real accelerator constants; gap-driven construction targets the cells where coverage is weakest; and LLM-assisted generation fills those cells under tight, schema-bound prompts.

Every STAFFML question is constructed for quantitative grounding and cognitive alignment with the competency model. The methodology draws on Evidence-Centered Design (ECD) (Mislevy et al., 2003) and Automatic Item Generation (AIG) (Gierl and Haladyna, 2013), prioritizing the structural properties that make questions valid assessments of ML systems expertise. The dominant question format throughout this paper is an open-response scenario with a worked napkin-math derivation; a small fraction of items are multiple-choice (four options, single correct index) and are used principally for screening, recall reinforcement, and quick warm-ups inside study sessions. Schema constraints on MCQ items are documented in Section 8.

7.1 Evidence-Centered Design

Every question begins with a specification of three ECD components: (1) the *competency claim*, what the question measures, defined by its topic $\times$ zone $\times$

track $\times$ level coordinates; (2) the *evidence*, what a correct answer demonstrates, mapped to Bloom’s cognitive operation; and (3) the *task features*, the scenario elements that elicit that evidence (hardware specs, failure mode, scale). The four-axis classification is specified before question construction, ensuring coverage is intentional rather than emergent. In the authoring schema, the scenario and the explicit question field are separated: the scenario establishes operational context, while the one-sentence question states the task the candidate must perform. This separation improves the practice UX and prevents candidates from inferring the intended answer format from the surrounding prose alone.

7.2 Hardware Grounding

Figure 6 summarizes the end-to-end construction path: gap analysis targets underfilled coverage cells, each item is assembled from taxonomy and hardware-grounded prompts, and only candidates that pass validation, deduplication, and math verification enter the corpus.

Each question is grounded in five components drawn from the schema:

- **Topic description** from the taxonomy: what concept to test
- **Zone description**: how to test it (which skills to exercise)
- **Track constraints**: which physics regime applies
- **Level expectations**: which Bloom’s cognitive operation to target
- **Hardware constants**: real specifications drawn from vendor datasheets (NVIDIA Corporation, 2022a,b; AMD, 2023; Arm Limited, 2020) (H100: 80 GB HBM3, 3.35 TB/s, 494 TFLOPS TF32 dense; MI300X: 192 GB HBM3, 5.3 TB/s; Orin: 275 TOPS INT8; Cortex-M4: 64 MHz, 256 KB SRAM)

Some items also include a visual task feature: a static diagram, timing trace, topology sketch, or memory layout rendered alongside the scenario. Visual items are used only when the diagram carries reasoning content beyond what prose alone conveys. To keep visual coverage principled rather than decorative, we maintain a curated catalog of visual archetypes that pair specific topics with the diagram class that best supports their reasoning act: ring/tree collective diagrams for collective communication, pipeline-bubble timelines for pipeline parallelism, KV-cache page layouts for cache management, queueing hockey-stick curves for tail-latency reasoning,

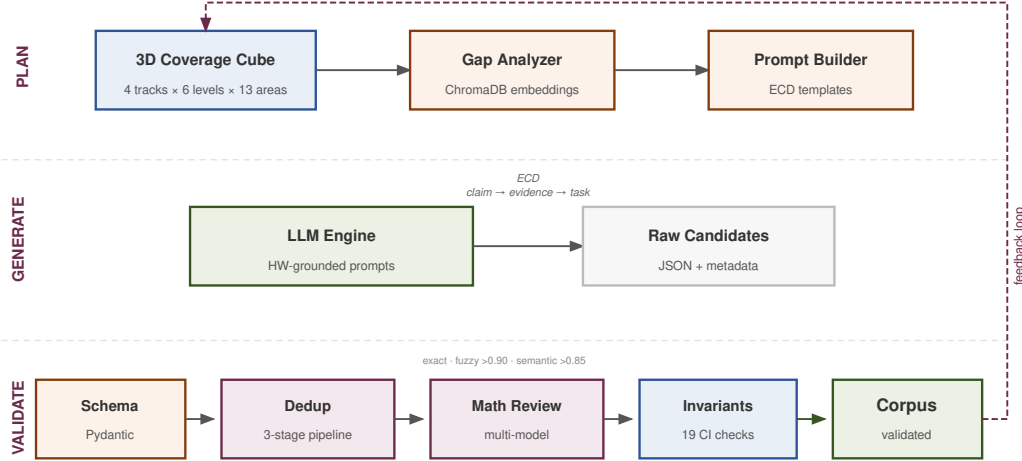


Figure 6. Item construction pipeline. Gap analysis identifies underfilled cells in the coverage space. Questions are constructed with hardware constants, Bloom’s descriptors, zone descriptions, and track constraints. Candidates pass through schema validation, multi-stage deduplication, multi-model math verification, and structural invariant checks before corpus integration.

memory-hierarchy data paths, interconnect topology placements, and sleep/wake duty-cycle and checkpoint/recovery timelines. A visual-coverage audit flags topics in the catalog that lack a visual exemplar, ensuring expansion of this archetype is gap-driven on the same footing as textual items. The visual asset is stored separately from the YAML so that the schema carries only metadata (kind, path, alt text, caption) while the website serves the figure as a static asset.

All questions are grounded in the content and formulas of the *Machine Learning Systems* textbook (Reddi, 2026b,a), which provides the hardware specifications, architectural principles, and quantitative frameworks that serve as the foundation for question construction. The hardware constants are drawn from the MLSysIM infrastructure modeling framework (Reddi, 2026c), ensuring that all questions reference the same specifications and that every napkin math answer can be independently verified against the framework. This prevents specification drift, a systematic error pattern where different questions cite different numbers for the same hardware, undermining the internal consistency that makes napkin math trustworthy. Hardware constants require maintenance as new accelerator generations ship; the methodology is designed to be timeless even as specific numbers evolve.

7.3 Gap-Driven Construction

Gap analysis identifies underfilled cells in the topic $\times$ zone $\times$ track space; construction prioritizes those cells. The coverage cube contains 233 semantically valid topic-track cells; summary statistics appear in Section 9. In practice, gap filling is staged: candidate items are first generated as drafts against explicit target specifications, then reviewed for scenario realism, answer alignment, napkin-math correctness, and user-facing clarity before promotion. This draft-first workflow is especially important for new item archetypes such as visual reasoning and incomplete-information questions, where schema validity alone is not enough to guarantee assessment quality.

The gap process is iterative rather than one-shot. After each draft batch, tooling recomputes the empirical coverage tensor across track, competency area, zone, level, and phase; ranks the weakest cells; and validates new candidates against schema, topic-track applicability, zone-level affinity, duplicate-question checks, visual-asset checks, and chain integrity. New items are added only when they improve a measured gap or repair a known quality weakness. This prevents the corpus from growing solely in volume and keeps expansion tied to the competency model.

A second, complementary loop monitors *distribution health*, not just coverage holes. Coverage asks whether every cell has at least some questions; balance asks whether the questions are distributed evenly enough that no axis collapses. We compute the

Gini coefficient and coefficient of variation over track $\times$ zone $\times$ level cross-products, weighting under-represented tracks (e.g., global, TinyML) and higher-order zones (e.g., realization, specification, mastery) more heavily so that imbalance triggers prioritized targeting. The planner is deterministic: given a corpus snapshot, it emits the next batch of weak cells to fill, which makes expansion auditable across releases. Coverage-driven and balance-driven targeting run as alternating passes, so the bank converges toward a portfolio that is both complete and proportioned to the competency model.

7.4 LLM-Assisted Generation

If STAFFML tests the systems reasoning that LLMs struggle with, how can LLMs generate the questions? LLMs struggle to reason about novel systems from first principles, but they effectively generate structured variations under fixed physical constraints when taxonomy, hardware constants, and task templates are supplied by the schema and MLSysIM. Generation is LLM-assisted; correctness is not taken on trust from any single model.

The value lies not in the drafting step but in the quality infrastructure that enforces physics grounding and validates each item before publication. Prompts fix the topic, zone, track, and level, inject the five grounding components above, and anchor scenarios in textbook material (Reddi, 2026b,a) so items stay tied to shared specifications rather than free-form model recall. The same coordinates and constants yield a reproducible structure across drafts; verification (Section 8) makes the bank safe to use.

The methodology treats LLMs as structured content generators rather than domain experts. Quantitative grounding comes from injected constants and schema constraints, not from training data. Two properties make the generated questions robust. First, every question requires reasoning based on specific hardware constants that the candidate must internalize rather than retrieve from a prompt. Second, the multi-model verification pipeline (Section 8) catches factual errors and arithmetic mistakes. The depth-chain structure (Section 3.5) organizes the corpus downstream of generation and is not itself a robustness mechanism.

The generator enforces a **validate-at-write contract**: every LLM-emitted YAML round-trips through the Pydantic schema before the file is written. Items that fail validation are dropped and never reach disk, removing an entire class of post-hoc repair work. The

generator also swaps in topic-class-specific prompt rules where general prompts produce systematic failures: a parallelism-specific variant, for example, forbids single-step bandwidth division, requires a concrete interconnect appropriate to the track (NVLink, InfiniBand, RoCE, PCIe, LoRa, or SPI), and demands a quantified synchronization or pipeline-bubble cost. With this variant in place, the same model produced 80.6% pass-on-first-judge versus 51% on the standard prompt, with roughly 1/9 as many API calls. Prompt specificity drives quality faster than budget.

The cumulative yield of this loop is concrete. The v0.1.0 release-readiness audits caught 462 `competency_area` drift fixes (zone-name substitutions, Bloom-verb substitutions, hallucinated underscore forms), 576 zone-Bloom mismatch reclassifications (e.g. `zone=recall` paired with `bloom=evaluate`), and reduced corpus-wide lint warnings from 1,308 to 0. The pipeline does not eliminate LLM-drafted errors; it makes them detectable, attributable, and bounded.

Every item undergoes the same quality path: schema validation, deduplication, and independent math verification (Section 8). Together, Figure 6 and Figure 7 illustrate the two halves of this continuous end-to-end generation and validation pipeline. A community contribution path for industry-sourced questions is planned to widen coverage of production failure modes.

8 Quality Assurance

Quality assurance is the path every draft question takes from candidate to published item, and it operates at three levels. At the schema level, a single LinkML definition guarantees that every consumer of the corpus interprets each field the same way. At the per-question level, a draft must pass schema validation, multi-stage deduplication, and multi-model math verification before it can be committed. At the corpus level, a 26-check invariant system enforces cross-file consistency that no single-question check can detect. The remaining subsections take each level in turn, then document the recurring failure modes that motivated several of the checks and the human and community review layer that backstops the automated pipeline. Figure 7 illustrates the per-question stages.

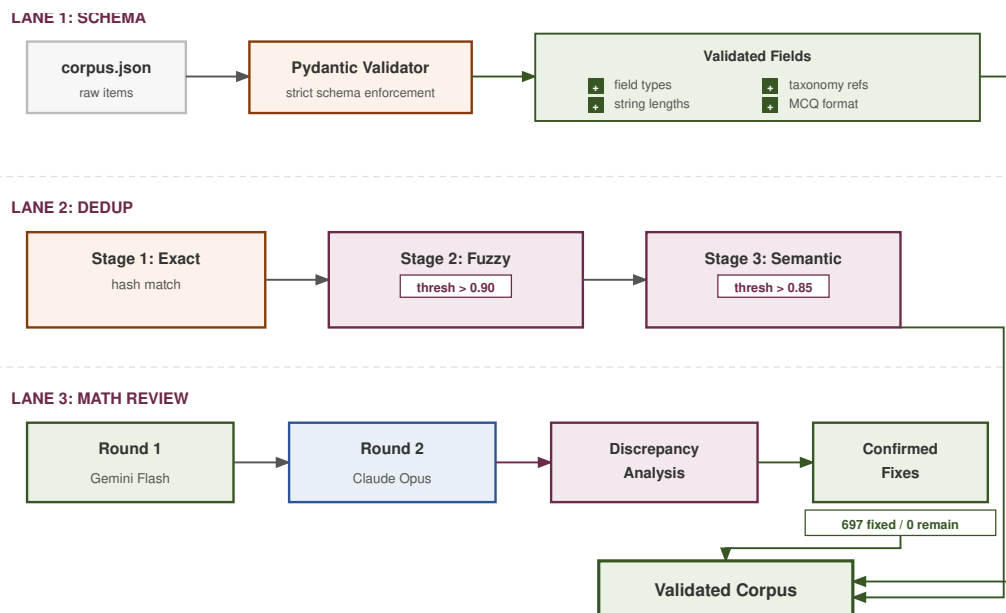


Figure 7. Multi-stage quality assurance pipeline. Candidate questions pass through schema validation (Pydantic), three-stage deduplication (exact, fuzzy, semantic), and multi-model math verification (Gemini Pro broad review + Claude Opus deep review on flagged items). Reviews are written as JSON reports rather than auto-applied; maintainers reconcile and apply confirmed corrections. Duplicates are archived, never deleted.

8.1 Schema as Single Source of Truth

The four-axis classification is useful only if every consumer of the corpus (the paper, the practice website, and the evaluation harness) agrees on the values each field can take. Four linked artefacts hold that agreement together: a LinkML schema as the canonical source of every type, enum, and constraint; Pydantic validation models that police the authoring and CI pipeline; a JSON Schema export consumed by the static corpus and offline tooling; and TypeScript bindings that drive the practice site. A topic identifier such as `kv-cache-management` is defined once in LinkML, validated by Pydantic at write time, exported to JSON Schema for the static bundle, and surfaced as a TypeScript literal in the website’s filter rail. We keep the three derived artefacts hand-written rather than fully code-generated and police them with a CI drift check that fails the build the moment any of them diverges from the LinkML source. The trade-off is deliberate: hand-written Pydantic validators can carry the cross-field rules that matter most (the zone–Bloom matrix, the visual asset-resolution check, the MCQ option-count rule) without forcing them through a code-generator’s escape hatches.

8.2 Schema Validation

Every question is validated against a strict Pydantic schema (kept in sync with the LinkML definition by a CI drift check) with both field-level and cross-question rules. Four classes of constraint, expanded in the v0.1.0 release-readiness push, now run as part of the validate-at-write contract:

- **Field constraints.** Minimum character lengths (scenario ≥ 50 , napkin math ≥ 10 and must contain a digit), explicit question prompts capped at 200 characters when present, MCQ must have exactly 4 options with the correct index in $[0, 3]$.
- **Visual constraints.** Visuals are a closed enum (SVG only); paths must be safe relative filenames in lowercase-dash form (no traversal, no uppercase, no underscores); alt text and captions have minimum lengths and captions are required, not optional. A separate model-level check requires that every declared visual resolve to a real file on disk, rejecting items that promise a diagram whose render silently fails.
- **Taxonomy constraints.** topic $\in \{87 \text{ curated IDs}\}$, zone $\in \{11 \text{ cognitive zones}\}$, track $\in \{5 \text{ tracks}\}$, level $\in \{L1 \dots L6+\}$; the Bloom level must be compatible with the zone (the hard zone–Bloom matrix; e.g. a recall zone cannot pair with the

evaluate Bloom level, since recall has no place for cognitive evaluation).

- **Uniqueness and linguistic constraints.** No duplicate IDs; no duplicate (track, title) pairs among published questions; no trailing quotes, markdown artifacts, or placeholder text.

A unit-test suite exercises each rejection path so that a regression in any constraint is caught at CI time rather than at the next corpus build.

Field coverage across the corpus is 100% for all required fields: `topic`, `zone`, `competency_area`, `napkin_math`, `common_mistake`, `realistic_solution`, and `bloom_level`. Optional fields, such as explicit question prompts and visuals, are validated when present and are included in the website export path so the practice interface can render them consistently. Under the multi-model math-verification pipeline (Section 8.4), 92.4% of published questions carry the validated flag.

8.3 Multi-Stage Deduplication

We employ a three-stage deduplication pipeline inspired by web-scale techniques (Broder, 1997):

1. **Exact match.** Lowercased, whitespace-stripped scenario comparison. Catches copy-paste duplicates.
2. **Fuzzy match.** `difflib.SequenceMatcher > 0.90` within the same (track, topic) groups. Catches minor rewordings.
3. **Semantic match.** Sentence-BERT (Reimers and Gurevych, 2019) cosine similarity > 0.85 via ChromaDB. Catches same-concept-different-words duplicates.

Confirmed duplicates are archived (status set to “archived”), never deleted, preserving the audit trail, a practice adopted from large-scale community assessment management (AnKing Team, 2024).

8.4 Math Verification

Napkin math is the soul of STAFFML and its most error-prone component. Hardware specifications change across generations, unit conversions cascade, and order-of-magnitude errors slip in easily. Verification combines automated checks with independent recomputation against a shared hardware-constants reference drawn from MLSysIM. As of the v0.1.0 release-readiness push, the pipeline is a three-stage loop in which the same frontier-model family plays three structurally distinct roles: a *generator* that drafts items under prompt rules and the *validate-at-write*

Table 6. Corpus summary. Key statistics of the STAFFML vault.

Metric	Value	Track	Questions
Published questions	9,521	Cloud	4,077 (42.8%)
Curated topics	87	Edge	2,093 (22.0%)
Competency areas	13	Mobile	1,826 (19.2%)
Ikigai zones	11	TinyML	1,208 (12.7%)
Mastery levels	6	Global	317 (3.3%)
Prerequisite edges	57		
Depth chains	843	Total	9,521
Schema	LinkML	Validated	92.4%

contract; a *judge* that scores drafts on rubric alignment and flags structural defects; and a *math reviewer* that recomputes arithmetic, specification, and unit issues from first principles in chunked, structured-output batches. A draft is accepted only when all three stages agree, in distinct prompt contexts and with distinct evidence requirements. Cross-stage agreement plays the role that cross-model agreement played in earlier designs and is the working empirical proxy for it. The pipeline is deliberately review-first rather than auto-correcting: every stage emits structured review reports without mutating the source YAML, so maintainers retain the final say on every applied correction. A future revision will add a second model family as a true cross-model check; recent work on LLM-as-judge evaluation (Zheng et al., 2023) supports cross-model agreement as a reliability improvement over single-model review. The corpus also allows community error reporting after release.

The full published set was checked under this loop; items with incorrect or inconsistent mathematics relative to the stated hardware scenario were revised or dropped before release. Because the loop is iterative, every release-candidate snapshot can be re-verified against the same constants reference, and items previously marked `validated` are re-evaluated when their grounding constants change. Readers should still treat any large bank as living software: constants evolve with new accelerators, and formal psychometric validation remains future work (Section 13).

8.5 LLM Failure Modes: A Taxonomy

Empirically, draft models exhibit a small set of recurring failure modes that single-pass review misses. Five surfaced repeatedly during the v0.1.0 release-readiness audits and each now has a dedicated guard: (i) **competency-area drift**, where the model substitutes a zone name, a Bloom verb, or a hallucinated un-

derscore form for one of the 13 canonical areas (over 60 surface patterns observed; the audit corrected 462 cases); (ii) **zone–Bloom mismatch**, where a question’s zone contradicts its Bloom level (for example a recall zone paired with the evaluate Bloom level, which is cognitively incoherent; 576 items reclassified); (iii) **visual drift**, where the YAML declares a visual but the rendered diagram never materialized because rendering crashed silently (caught by structured per-ID failure logging plus the asset-resolution validator); (iv) **trivial-division framings** at the highest levels, where a single-step `payload / bandwidth` arithmetic is dressed as a Staff-level question (rejected by topic-class-specific prompt variants that demand a quantified synchronization or pipeline-bubble cost); and (v) **wrong-track scenarios**, where a microcontroller appears in a Cloud question or vice versa (caught by topic-track applicability invariants). In addition, models still confuse accelerator specifications between product lines, drop factors in collective-communication or roofline calculations, and swap unit prefixes so digits look right but magnitude is wrong; cross-stage recomputation against a single constants source catches these residual classes. Together, the five named guards plus residual cross-checks reduced corpus-wide lint warnings from 1,308 to 0.

8.6 Structural and Semantic Invariants

Beyond per-question validation, STAFFML enforces 26 strict invariants across the dataset. While some are standard referential integrity checks (e.g., ensuring the prerequisite graph is acyclic and all foreign keys resolve), the system’s true value lies in enforcing domain-specific semantic and physics constraints:

- **Physics applicability.** A question’s topic-track pair must strictly adhere to the physics exclusion matrix (e.g., an LLM generating an RDMA question for a TinyML track is immediately rejected at commit-time).
- **Cognitive affinity.** A question’s zone must be cognitively compatible with its Bloom’s level. For example, a question assigned to the “Evaluation” zone is structurally rejected if its difficulty is tagged as L1 (Remember) or L2 (Understand).
- **Chain progression.** Questions linked in a depth chain must strictly monotonically increase in Bloom’s level; a chain cannot regress from L4 back to L2.

- **Coverage health.** The distribution cannot collapse: no single topic may exceed 10% of the corpus, and all defined zones must maintain a minimum population threshold to prevent model drift toward easy-to-generate topics.

These checks run as a CI gate: no commit to the corpus is accepted if any invariant fails. A traditional LLM validation pass might miss that an L1 Evaluation question is an oxymoron, but the invariant system guarantees the structural and cognitive coherence of the entire 9,521-question bank. For release planning, we also compute a gap-analysis scorecard over the four primary axes and maintain a repair backlog that separates hard errors (e.g., scenario-solution mismatch) from advisory warnings (e.g., a draft item whose zone lies outside the usual level of affinity).

Each invariant is paired with deterministic repair tooling. When an invariant fires, the corresponding repair pass applies bounded, auditable fixes that the maintainer can review before commit: zone–Bloom mismatches are relabeled against the canonical zone–Bloom mapping (the audit corrected 576 items in one pass), `competency_area` drift is resolved against the canonical area enum (462 cases corrected), and visual-render failures are logged per-ID so the asset and its declaring YAML can be reconciled together. Invariant violations, therefore, become operational signals, not terminal errors, and the path from a failed gate to a green corpus is a bounded set of repair passes rather than a manual triage exercise.

8.7 Human and Community Review

Automated validation is necessary but not sufficient for assessment quality. STAFFML therefore separates three signals: automated validation, maintainer review, and community feedback. The schema records human-review status independently from LLM validation stamps; the practice interface also lets users mark whether a question was useful, too easy or too hard, and whether they believe the prompt, answer, and math are verified or disputed. These community signals are not treated as ground truth, but they provide a triage layer: questions with many disputes or low usefulness scores are candidates for human review, while questions with repeated verification and good answer outcomes become stronger candidates for promotion into curated study paths.

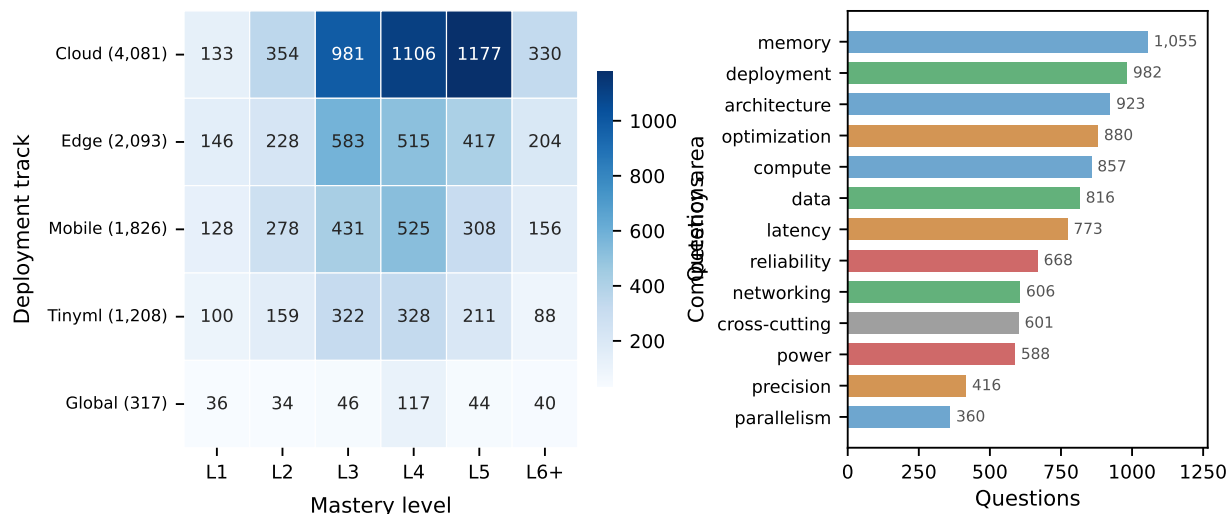


Figure 8. Track $\times$ Level heatmap. The left panel shows the track $\times$ mastery-level count grid; the right panel shows question counts by competency area. Cloud L3–L5 cells are densest, reflecting industry emphasis on mid-to-senior systems reasoning.

9 Coverage Analysis

Construction and quality assurance describe how individual questions enter the corpus; coverage analysis is the corresponding aggregate view. Once the per-question gates have done their job, the open question is whether the resulting bank exercises every cell of the four-axis space the assessment claims to cover. This section examines how questions are distributed across topic, zone, track, and level, highlighting where the corpus is strong and where adding more items would improve balance.²

9.1 Track Distribution

The track distribution reflects both industry demand and regime breadth. Cloud is the densest track, consistent with datacenter LLM serving and distributed training. Edge, mobile, and TinyML carry substantial mass so on-device constraints appear throughout the bank. The global track holds cross-platform conceptual items. Table 6 reports the exact distribution. Figure 8 shows the same corpus as a track $\times$ mastery-level heatmap: Cloud questions at L3–L5 form the densest cells, consistent with industry demand for mid-to-senior backend systems engineers. The right panel of the same figure tallies questions by competency area: compute, deployment, and cross-

cutting carry the largest mass because their topics are universal across all four tracks (Table 8) and therefore populate the most cells of the topic $\times$ track grid; parallelism sits at the bottom because most of its topics are cloud-only and thus contribute exclusions on three of the four tracks.

9.2 Zone Distribution

Figure 9 gives per-zone counts across the eleven cognitive zones. Compound zones (diagnosis, fluency, evaluation) dominate the corpus, which is intentional: they most sharply differentiate Staff-level candidates. Ten of the eleven zones meet or exceed the 500-question authoring floor, with diagnosis (1,536), fluency (1,157), and evaluation (1,088) leading the distribution; only realization sits below the floor and is targeted for growth in the next portfolio-balance pass (Section 7).

9.3 Level and Bloom’s Distribution

The Bloom rollup, derived from each item’s primary cognitive zone, places Analyze (2,330 questions, 24.5%) at the top, followed by Apply (1,402, 14.7%), Understand (1,157, 12.2%), Evaluate (1,088, 11.4%), Remember (901, 9.5%), and Create (780, 8.2%). The remaining items sit in compound zones (mastery, optimization, realization) that span multiple Bloom levels and are intentionally not folded into a single Bloom bucket; they read as Staff-level integrative reasoning rather than as a single cognitive verb. The

²Numeric counts in this section are emitted by the same export pipeline that builds the practice website’s API, so the paper and the site agree by construction.

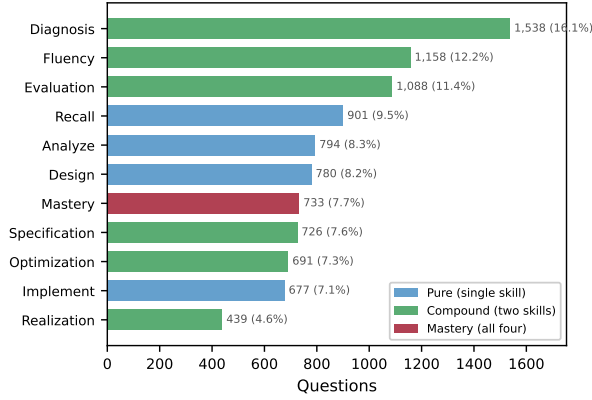


Figure 9. Zone distribution. Compound zones (diagnosis, fluency, evaluation) dominate by design: they most sharply differentiate Staff-level candidates.

dominance of Analyze, paired with the substantial mass in those multi-level compound zones, reflects the higher-order reasoning required at the Staff level. Figure 11 complements these category totals by stacking item formats (conceptual, calculation, design, diagnosis, tradeoff, optimization) across Bloom levels.

9.4 Multi-Axis Coverage

Figures 10 and 11 summarize how zones, Bloom levels, and item formats line up under the authoring priors. The zone $\times$ level heatmap shows each cognitive zone clustering at its intended difficulty: recall peaks at L1–L2, fluency anchors at L3, diagnosis centers on L3–L4, and mastery is confined to L6+. The chart shows the same story at the item level: conceptual prompts dominate junior levels, calculation peaks at L3, and open-ended design dominates L5–L6+.

The three-dimensional coverage space of topic $\times$ zone $\times$ track contains $87 \times 11 \times 4 = 3,828$ cells (excluding the small “global” track). Not all cells are semantically meaningful: a TinyML question about InfiniBand fabric topology, or a cloud question about MCU duty cycling, has no physical substrate to reason about. We define an *applicability matrix* that excludes 83 topic–track pairs where the underlying concept has no physical realization on that hardware tier. Each exclusion carries a deployment-feasibility justification grounded in the relevant physics (e.g., “RDMA requires InfiniBand/RoCE NICs; edge uses Ethernet/WiFi”). The remaining 233 applicable pairs yield 2,563 cells (233×11 zones).

Table 7 tabulates every topic–track pair; Table 8 aggregates those exclusions into summary counts. Cloud-only topics (11) involve datacenter-scale infrastructure such as pipeline parallelism, RDMA

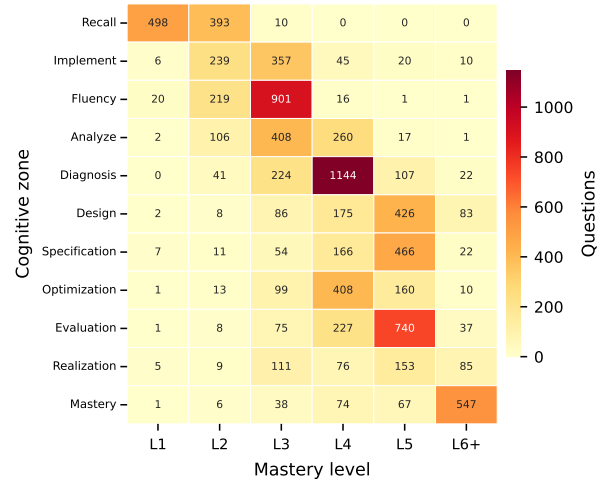


Figure 10. Zone $\times$ Level heatmap. Each cognitive zone concentrates at its affinity Bloom levels.

transport, and container orchestration, all of which require multi-device interconnects absent from edge, mobile, and TinyML platforms. Conversely, MCU compute constraints, OTA firmware updates, and duty cycling exclude cloud because their engineering challenges (KB-scale memory, battery harvesting, real-time deadlines) have no analogue in datacenter environments. The TinyML track carries the most per-track exclusions (39 of 79 topics excluded), reflecting the fundamental constraint that microcontrollers with 256 KB SRAM cannot instantiate concepts like KV-cache management, mixture-of-experts routing, or activation checkpointing.

Reading the grid, checkmarks indicate a meaningful physical substrate for interview questions and crosses mark physics-grounded exclusions. The visual pattern aligns with Table 8: parallelism and networking skew cloud-only on constrained tiers, while precision and cross-cutting topics are largely universal. The TinyML column carries the most exclusions, reflecting that microcontrollers with 256 KB SRAM cannot instantiate concepts requiring GB-scale memory.

10 Example Questions

The coverage analysis showed how the four axes carve the corpus into cells; this section makes those cells concrete by walking through actual questions. The point is not to display many questions but to demonstrate that each axis is doing real work; zone changes the cognitive act, level changes the depth,

Table 7. Applicability matrix. Each row is a topic; the four marks are Cloud, Edge, Mobile, and TinyML. **Bold** rows are category headers. Read across a row for a topic’s tracks; read each of the three topic columns top to bottom (rows do not align across columns). ✓: applicable; ✗: excluded. Of 87×4 pairs, 233 are applicable and 83 excluded.

Topic	Cloud	Edge	Mobile	TinyML	Topic	Cloud	Edge	Mobile	TinyML	Topic	Cloud	Edge	Mobile	TinyML
Architecture (8)					Deployment (9)					Optimization (7)				
Attention Scaling & Variants	✓	✓	✓	✓	A/B & Rollout Strategies	✓	✓	✓	✗	Graph Compilation & Optimization	✓	✓	✓	✓
CNN Efficient Design	✓	✓	✓	✓	Compound AI Systems	✓	✓	✓	✗	IO-Aware Attention	✓	✗	✗	✗
Encoder-Decoder Tradeoffs	✓	✓	✓	✗	Container & Cluster Orchestration	✓	✗	✗	✗	Kernel & Operator Fusion	✓	✓	✓	✗
Mixture of Experts	✓	✗	✗	✗	Disaggregated Serving	✓	✓	✓	✓	Knowledge Distillation	✓	✓	✓	✓
Model Size Estimation	✓	✓	✓	✓	MLOps Lifecycle	✓	✓	✓	✗	Operator Scheduling	✓	✓	✓	✓
Neural Architecture Search	✓	✓	✓	✓	Model Adaptation Systems	✓	✓	✓	✓	Pruning & Sparsity	✓	✓	✓	✓
Recommendation Systems Eng.	✓	✓	✓	✓	Model Format & Runtime Conversion	✗	✓	✓	✓	Speculative Decoding	✓	✓	✓	✗
Transformer Systems Cost	✓	✓	✓	✓	Model Serving Infrastructure	✓	✓	✓	✗	Parallelism (7)				
Compute (7)					OTA & Firmware Updates	✗	✓	✗	✓	3D Parallelism	✓	✓	✗	✗
Accelerator Comparison	✓	✓	✓	✓	Latency (6)					Comm./Compute Overlap	✓	✓	✓	✓
Chiplet Architecture	✓	✓	✓	✓	Batching Strategies	✓	✓	✗	✗	Data Parallelism	✓	✓	✗	✗
Compute Cost Estimation	✓	✓	✓	✓	Latency Decomposition	✓	✓	✓	✓	Gradient Synchronization	✓	✗	✗	✗
GPU Compute Architecture	✓	✓	✗	✗	Profiling & Bottleneck Analysis	✓	✓	✓	✓	Model & Tensor Parallelism	✓	✗	✗	✗
MCU Compute Constraints	✗	✗	✗	✓	Queueing Theory	✓	✓	✗	✗	Pipeline Parallelism	✓	✗	✗	✗
Roofline Analysis	✓	✓	✓	✓	Real-Time Deadlines	✗	✓	✓	✓	Scheduling & Resource Management	✓	✗	✗	✗
Systolic Arrays & Dataflow	✓	✓	✓	✓	Tail Latency & SLAs	✓	✓	✗	✗	Power (5)				
Cross-Cutting (8)					Memory (8)					Datacenter Efficiency	✓	✗	✗	✗
Autograd & Comp. Graphs	✓	✓	✓	✓	Activation Memory & Checkpointing	✓	✓	✗	✗	Duty Cycling & Energy Harvesting	✗	✓	✓	✓
Differential Privacy	✓	✓	✓	✗	DMA & Data Movement	✓	✓	✓	✓	Energy Per Operation	✓	✓	✓	✓
Fairness & Bias Evaluation	✓	✓	✓	✗	KV-Cache Management	✓	✗	✓	✗	Power Budgeting	✓	✓	✓	✓
Federated Learning	✓	✓	✓	✓	Memory Hierarchy Design	✓	✓	✓	✓	Thermal Management	✓	✓	✓	✗
Responsible AI & Governance	✓	✓	✓	✓	Memory Pressure Management	✓	✓	✓	✗	Precision (3)				
Software Portability	✓	✓	✓	✓	Memory-Mapped Inference	✓	✓	✓	✗	Extreme Quantization	✓	✓	✓	✓
Sustainability & Carbon Acct.	✓	✓	✓	✓	Tensor Arena Planning	✓	✓	✗	✓	Mixed-Precision Training & Inference	✓	✓	✓	✓
TCO & Cost Modeling	✓	✓	✓	✓	VRAM Budgeting	✓	✓	✓	✓	Quantization Fundamentals	✓	✓	✓	✓
Data (7)					Networking (6)					Reliability (6)				
Data Efficiency & Selection	✓	✓	✓	✗	Collective Communication	✓	✓	✓	✓	Adversarial Robustness & Security	✓	✓	✓	✓
Data Pipeline Engineering	✓	✓	✓	✓	Congestion & Flow Control	✓	✗	✗	✗	Distribution & Drift Detection	✓	✓	✓	✗
Data Quality & Validation	✓	✓	✓	✗	Interconnect Topology	✓	✗	✓	✗	Fault Tolerance & Checkpointing	✓	✓	✓	✗
Dataset Curation & Labeling	✓	✓	✓	✓	Load Balancing & Routing	✓	✓	✗	✗	Graceful Degradation	✓	✓	✓	✓
Feature Store & Feature Engineering	✓	✗	✗	✗	Network Bandwidth Bottlenecks	✓	✗	✓	✗	Monitoring & Observability	✓	✓	✓	✓
Storage Format & Selection	✓	✓	✗	✗	RDMA & High-Performance Transport	✓	✗	✗	✗	Safety & Certification	✗	✓	✓	✓
Streaming & Real-Time Ingestion	✓	✓	✓	✓										

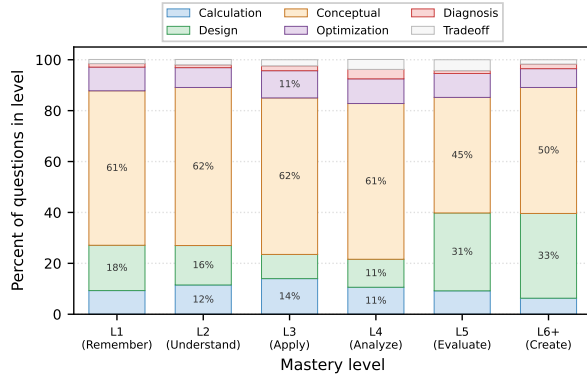


Figure 11. Question format distribution by Bloom's level. Conceptual questions dominate L1–L2, calculation peaks at L3, design dominates L5–L6+.

track changes the physics, and the prose-or-diagram format is itself a signal carrier.

We present seven examples, each varying one of these dimensions at a time. The first five hold track constant (cloud) and progress through five zones (fluency, diagnosis, specification, evaluation, mastery) at increasing levels (L3–L6+), so the reader can observe how cognitive demand changes independently of the physics regime. The sixth holds topic constant (quantization) and varies track across cloud, edge, and TinyML, revealing how the same concept transforms under different hardware constraints. The seventh is a visual-reasoning item whose diagram conveys the reasoning, illustrating a question form that prose alone cannot convey.

10.1 Fluency: KV-Cache Sizing

Fluency tests whether a candidate can perform napkin math from memory, recalling both the formula and the hardware specifications, then executing the arithmetic. It is the signature skill tested in Staff-level phone screens.

vram-budgeting
fluency • cloud • L3

Scenario: You need to increase your LLM's context window from 4K to 128K tokens. The model weights fit perfectly in your 80 GB VRAM. What hidden memory cost will cause your node to OOM during generation?

Common mistake: "Activations will use more memory." Activations matter during training, but during inference the dominant hidden cost is something else entirely.

Napkin math: For Llama-70B (80 layers, 8 KV-heads via grouped-query attention ([Ainslie et al., 2023](#)),

Table 8. Applicability matrix summary. Topic–track exclusions grouped by deployment pattern. Each row reports the number of topics in the pattern and the number of topic–track exclusion pairs the pattern contributes; the columns sum to the totals at the bottom. Counts cover the original 79-topic v0 taxonomy on which the matrix is curated.

Pattern	Topics	Excl.	Example topics
Universal (4 tracks)	33	0	roofline, quantization, distillation
Cloud-only	11	33	TP/PP, RDMA, MoE, K8s
TinyML excluded only	16	16	KV-cache, kernel fusion, batching
No-cloud (device only)	4	4	duty cycling, OTA, safety cert
TinyML only	1	3	MCU compute constraints
Selective (mixed)	14	27	GPU arch, flash-attn, batching
Total	79	83	233 applicable pairs → 2,563 cells

head_dim=128) at 128K tokens in FP16:

$$\underbrace{2}_{\text{K,V}} \times \underbrace{80}_{\text{layers}} \times \underbrace{8}_{\text{heads}} \times \underbrace{128}_{\text{dim}} \times \underbrace{128\text{K}}_{\text{tokens}} \times \underbrace{2}_{\text{bytes}} \approx \mathbf{41.9\text{ GB}}$$

Combined with 140 GB of weights, one 128K-context request requires ~182 GB. This spills over an 80 GB NVIDIA H100 (requiring 3 GPUs) but fits entirely on a single 192 GB AMD MI300X.

Answer: The KV-cache. It grows linearly with sequence length and, at 128K context, can exceed the model weights themselves.

This is a fluency question. The candidate must recall the KV-cache formula and execute the arithmetic from memory, producing a specific number. No architectural design is required; no tradeoff analysis is asked for. The cognitive act is rapid quantitative calculation, the signature skill of the fluency zone.

Timeless principle: KV-cache memory scales linearly with sequence length and quadratically with attention head count. The specific hardware capacities change (e.g., 80 GB vs. 192 GB); the scaling law does not.

10.2 Diagnosis: MFU Collapse

Diagnosis tests whether a candidate can identify root causes from observable symptoms. The candidate must recall hardware specifications and reason about which bottleneck explains the observed behavior. This is the "3 AM on-call" skill.

Scenario: Your production LLM serving cluster reports an 8% MFU during the decode phase at batch size $B \approx 16$. The H100s are rated at 494 TFLOPS TF32 dense. Your team assumes the GPUs are underclocked or driver-limited and pages the platform team. Diagnose the actual root cause.

Common mistake: Blaming hardware or driver issues. The decode phase of autoregressive generation is inherently memory-bandwidth-bound at small-to-moderate batch sizes, not compute-bound.

Napkin math: Per decode step, the model reads weights ($2 \times 70B = 140GB$ in FP16) once and applies them to B in-flight tokens, performing $B \times 140$ GFLOPs of compute against the same 140 GB of HBM traffic. At $B = 16$, arithmetic intensity is $16 \times 140/140GB = 16$ FLOP/Byte, still far below the H100's ridge point of ~ 148 FLOPs/Byte (dense). The memory-bound ceiling is $3.35TB/s \times 16 \approx 54$ TFLOPS, or $\sim 11\%$ of the 494 TFLOPS peak; the observed 8% MFU is therefore $\sim 73\%$ of that ceiling, close to optimal for this regime and not a hardware deficit.

Answer: The decode phase is memory-bandwidth-bound. MFU measures compute utilization, but the bottleneck is data movement. Because arithmetic intensity scales linearly with batch size, the only structural fix is to raise B (continuous batching, larger client concurrency); shipping bigger GPUs cannot help when the kernel is already starved for bytes.

This is a diagnosis question. The candidate must recall hardware specifications (H100 bandwidth, ridge point) and analyze why those specs explain the observed symptom (low MFU). The cognitive act is “identify and explain a failure from first principles.”

Timeless principle: When arithmetic intensity falls below the hardware ridge point, adding compute does not help. Workloads shift from compute-bound to memory-bound as batch size decreases.

10.3 Specification: Latency-Based Serving

Specification tests whether a candidate can translate quantitative constraints into a concrete architecture. The candidate must recall hardware capabilities and design a system that satisfies a hard performance requirement. This is the core skill of system design interviews.

Scenario: Your team must serve a 70B-parameter LLM with P99 time-to-first-token (TTFT) under 200 ms. The model is deployed on H100 GPUs with NVLink. Specify the minimum serving configuration.

Common mistake: Focusing only on model parallelism degree while ignoring the prefill compute budget.

Napkin math: Prefill FLOPs for a 2,000-token prompt:

$$\underbrace{2}_{\text{MACs}} \times \underbrace{70B}_{\text{params}} \times \underbrace{2,000}_{\text{tokens}} = 280 \text{ TFLOPs}$$

A single H100 at 494 TFLOPS TF32 dense completes this in $280/494 \approx 567$ ms, far exceeding the 200 ms SLO. With 4-way tensor parallelism, prefill drops to ~ 142 ms, fitting the budget with margin for scheduling overhead. Model weights (140 GB FP16) require 2 GPUs minimum for capacity. The configuration is $4 \times$ H100 with tensor parallelism, continuous batching, and chunked prefill.

Answer: 4-way tensor parallelism across H100s with NVLink. The constraint is prefill compute, not memory capacity: even though the model fits on 2 GPUs, the latency SLO demands additional parallelism.

This is a specification question. The candidate must recall hardware capabilities (H100 TFLOPS, NVLink bandwidth) and design a serving architecture that satisfies a quantitative constraint (P99 TTFT < 200 ms). The cognitive act is “translate constraints into architecture.”

10.4 Evaluation: The Communication Tax

Evaluation tests whether a candidate can weigh competing architectures and justify a choice with quantitative reasoning. The candidate must recall hardware capabilities and analyze which design keeps the most expensive communication off the critical path. This is the core skill of architecture review.

Scenario: You are training a 175B-parameter model across 1,024 GPUs. Nodes are connected by 400 Gb/s InfiniBand; GPUs within a node share 3.2 TB/s NVLink. Evaluate whether pipeline parallelism (PP) or tensor parallelism (TP) should cross the inter-node fabric to minimize the communication tax.

Common mistake: Treating all bandwidth as equivalent and applying TP across nodes to avoid pipeline bubbles, ignoring the latency penalty of InfiniBand AllReduce on the critical path.

Napkin math: TP requires a blocking AllReduce for every layer ($2 \times$ activation size). Over 400 Gb/s InfiniBand, these frequent synchronizations dominate step time and collapse scaling efficiency; over 3.2 TB/s NVLink they are easily absorbed. PP requires only point-to-point transfer of activations at stage boundaries, a fraction of the bandwidth that 400 Gb/s InfiniBand can comfortably handle.

Answer: TP stays within the node (over NVLink); PP crosses nodes (over InfiniBand). The communication tax dictates the parallelism topology: high-frequency, latency-sensitive synchronizations belong on the high-bandwidth domain, while lower-frequency, bandwidth-tolerant transfers can span the slower fabric.

This is an evaluation question. The candidate must recall the bandwidth and latency gap between NVLink and InfiniBand and analyze which parallelism strategy keeps the blocking synchronizations on the fast fabric. The cognitive act is “weigh competing designs against fixed constraints and justify the choice.”

Timeless principle: The fastest interconnect in a hierarchy absorbs the most frequent communication. As long as intra-node bandwidth exceeds inter-node bandwidth, tensor-parallel AllReduce belongs inside the node regardless of which accelerator generation is in use.

10.5 Mastery: Continuous Batching Stall

Mastery requires all four skills simultaneously: recalling specs, analyzing the failure mode, designing an architectural fix, and sizing the solution with napkin math. These questions simulate the complete Staff-level interview loop.

batching-strategies
mastery • cloud • L6+

Scenario: You implement continuous batching for an LLM endpoint. Under low load, TTFT is 100 ms. Under high load, throughput triples, but TTFT spikes to over 4 seconds even though token generation speed remains fast. Diagnose the root cause, design a fix, and estimate the resource cost of your solution.

Common mistake: Assuming that because continuous batching optimizes throughput, it automatically guarantees low latency for incoming requests.

Napkin math: A 2,000-token prefill on a 70B model: $2 \times 70B \times 2,000 = 280$ TFLOPs. An H100 at 494 TFLOPS dense needs ~ 567 ms. If 4 users send prompts simultaneously, the 4th waits over 1.5 seconds before prefill even begins.

Answer: Prefill starvation. The scheduler must chunk or defer prefills to prevent blocking the decode phase, but aggressive deferral under high load creates a queue that destroys TTFT. The fix is chunked prefill with priority scheduling (Agrawal et al., 2024), but this requires overprovisioning compute by $\sim 30\%$ to absorb the prefill bursts without queuing.

This is a mastery question. The candidate must recall hardware specs (H100 TFLOPS), analyze the scheduling conflict (prefill blocks decode), design a solution (chunked prefill with priority), and execute the napkin math to size the required overprovisioning. All four skills are exercised in a single question.

Timeless principle: In multiplexed serving, prefill and decode compete for the same compute; any

scheduling policy must bound prefill latency to prevent decode starvation.

10.6 Cross-Track Quantization

The preceding four examples held track constant to isolate zone differences. We now hold topic constant (quantization) and vary the track, showing how the same concept demands different reasoning under different physics regimes:

- **Cloud** (L4, evaluation). “GPTQ (Frantar et al., 2023) vs. AWQ (Lin et al., 2024) for serving a 70B model on an Intel Gaudi 3 cluster. Which quantization scheme yields higher throughput at FP16-equivalent perplexity? Justify with arithmetic intensity analysis.” The candidate evaluates two compression strategies against datacenter-scale metrics (tokens/second/dollar).
- **Edge** (L3, fluency). “Quantize a YOLOv8 detection model from FP32 to INT8 for Orin deployment. Calculate the memory savings and estimate the latency improvement given Orin’s INT8 TOPS vs. FP32 TFLOPS.” The candidate performs napkin math within a thermal and real-time envelope.
- **TinyML** (L5, specification). “Your 256 KB MCU must run a keyword spotting model. The FP32 model is 800 KB. Specify a quantization strategy (INT8, INT4, or binary) that fits in SRAM while maintaining $>90\%$ accuracy. Justify each precision choice for weights vs. activations.” The candidate designs a mixed-precision scheme under extreme memory constraints.

All three questions test quantization knowledge, but the physics regime determines the constraints, the formulas, and the engineering tradeoffs. A candidate who can answer the cloud question may struggle with the TinyML variant, because the reasoning transfers only partially across regimes.

10.7 Visual Reasoning: Pipeline Bubbles

The preceding examples are pure prose. A subset of STAFFML items pair the scenario with a diagram when the reasoning depends on reading a picture rather than parsing words. The example below tests pipeline-parallel scheduling: the candidate must read the timeline, count active vs. idle stage-time cells, and recover the bubble fraction.

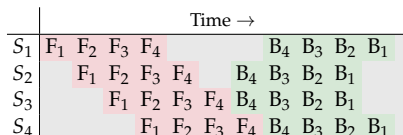
This is a realization-zone question, but the cognitive act is specifically diagram-driven. The same arithmetic posed without the timeline tests memory

of the bubble formula; posed with the timeline, it tests whether the candidate can derive the formula by reading the schedule. Visual items are kept to cases like this, where the picture carries reasoning content that prose cannot, rather than as decoration.

Together, these seven questions show the classification doing different kinds of work along different axes. Holding track constant (the first five examples) reveals qualitatively different cognitive acts: fluency tests whether you can compute a number, diagnosis whether you can explain an observation, specification whether you can translate constraints into architecture, and mastery whether you can synthesize a complete solution. Holding topic constant (the cross-track quantization example) reveals that the same concept demands different formulas, constraints, and engineering tradeoffs depending on the physics regime. The visual example then shows that the question *format* is itself a signal carrier: when the reasoning act depends on reading a diagram, the prose-only variant tests a different skill.

pipeline-parallelism
realization • cloud • L4

Scenario: A 4-stage GPipe (Huang et al., 2019) pipeline runs $m = 4$ micro-batches per minibatch. The figure below traces stages S_1 – S_4 across time, with forward (F) and backward (B) cells per micro-batch and idle bubbles in gray. Read the diagram and compute the bubble fraction; then estimate how it changes if you increase the micro-batch count to $m = 16$.



Common mistake: Counting only the leading bubble (the staircase fill on the left) and ignoring the trailing bubble (the drain after the last micro-batch). Both ends contribute, and at small m they dominate the schedule.

Napkin math: With p stages and m micro-batches, each stage runs for $2m$ useful steps (forward + backward) but the schedule spans $2m + (p - 1)$ steps end to end, so the bubble fraction is $(p - 1) / (m + p - 1)$. Reading the figure: $p = 4$, $m = 4$, bubble fraction = $3/7 \approx 43\%$. At $m = 16$, bubble fraction = $3/19 \approx 16\%$.

Answer: The bubble fraction at $m = 4$ is $\sim 43\%$; raising m to 16 cuts it to $\sim 16\%$. The diagram makes the structural cost of pipeline parallelism visible in a way that prose obscures: the staircase ends do not shrink, only the productive middle grows.

Future work includes questions that provide partial information, requiring candidates to identify

what additional data they need, a skill that tests meta-cognitive reasoning about systems diagnostics. Such incomplete-information questions would assess whether candidates know what they do not know, a hallmark of Staff-level engineering judgment.

11 Practical Applications

The example questions show what individual cells look like; the natural next question is what an interviewer or candidate should do with the bank. The classification is useful only if it changes how interview loops are assembled and how practice sessions are shaped, so this section turns the four-axis structure into concrete recipes for both audiences. The zone model maps directly onto interview loop design. A typical four-round interview should cover at least four zones to assess breadth of systems reasoning:

- **Phone screen.** Fluency questions. Can the candidate rapidly estimate performance bounds from memory? This is the fastest signal for mechanical sympathy.
- **System design.** Specification or Design questions. Can the candidate architect a system under quantitative constraints?
- **Debugging.** Diagnosis questions. Can the candidate root-cause a failure from observable symptoms and hardware specifications?
- **Architecture review.** Evaluation or Mastery questions. Can the candidate analyze tradeoffs and justify design decisions with quantitative reasoning?

A “strong hire” signal at Staff level is consistent strength in compound zones, especially diagnosis and evaluation. A “no hire” signal is strong recall but weakness across all compound zones: the profile of a memorizer rather than a reasoner. No interview loop should test only pure zones, as these fail to exercise the integrative reasoning that distinguishes Staff-level engineers from mid-level ones.

The full zone-by-zone rationale (including why realization is rare and why we retain all eleven tags) is in Section 3. For *hiring loops*, the actionable recipe is: center rounds on the five compound zones and mastery, use pure zones and realization for screening, warm-up, and study material rather than as the sole bar for a Staff+ decision, and read Section 3 when you need the cognitive definition of each zone.

The practice interface exposes the four-axis structure as filters so candidates can shape a session

Table 9. Interview preparation landscape. Existing platforms optimize for algorithmic coding or conceptual systems knowledge; STAFFML is the first to combine quantitative rigor with ML systems focus and a structured taxonomy.

Platform	Items	Systems	Quantitative	HW-Grounded	Cognitive Model
LeetCode	2,800+	–	✓	–	Easy / Med / Hard
NeetCode	150	–	✓	–	Pattern-based
Grind 75	169	–	✓	–	Frequency rank
Huyen (DMLS)	Conceptual	✓	–	–	–
STAFFML	9,521	✓	✓	✓	11 zones × 6 levels

against their target role: a TinyML candidate can isolate the TinyML track, an L5 candidate can restrict to L4–L6+ items, and a candidate preparing for a sequenced study path can switch to a chains-only view that surfaces the 843 depth chains while hiding standalone questions. Visual-reasoning items are likewise filterable so candidates can practice diagram-driven reasoning specifically. These filters are a thin presentation layer over the same four-axis schema described in Sections 3 to 6; no taxonomy is duplicated for the UI.

Figure 2, introduced in Section 1 as the artifact’s big picture, shows a pre-reveal practice frame in detail. The left rail holds axis filters (track, level, competency, zone) and session toggles; the center holds the card with chain position, task callout, scenario, napkin-math input, and *Reveal Answer*, and after reveal presents the model answer and self-rating (Skip / Wrong / Partial / Nailed It in the live UI); the right rail is the tools stack (hardware reference, napkin calculator, Ask Interviewer). The interface is intentionally low-friction: each session is a sequence of in-place reveals, and session-level pool counts stay client-side except as described in Section 11. Self-grade signals are written back to the corpus only as anonymous aggregate counters used for the community-feedback triage in Section 8.

12 Related Work

STAFFML sits at the intersection of four bodies of work, one per subsection that follows. Interview preparation platforms supply the prior art for assessment at scale and define the gap that motivates the present work. LLM engineering benchmarks share the format of fixed-answer technical evaluation but target agent capability rather than human judgment, and they frame how STAFFML can serve as a mechanical-sympathy benchmark. ML systems research supplies the technical content the questions are grounded in. Educational measurement theory

grounds the assessment methodology and informs the four-axis classification, depth chains, and validation pipeline.

12.1 Interview Platforms

LeetCode (LeetCode Inc., 2024) hosts 2,800+ algorithmic coding problems with community-driven difficulty calibration and is the dominant platform for software engineering interview preparation. NeetCode (NeetCode, 2024) curates 150 problems organized by algorithmic pattern with explicit difficulty progression. Grind 75 (Tay, 2024) ranks 169 problems by interview frequency and pattern coverage. All three platforms excel at algorithmic coding assessment but provide no coverage of systems reasoning, hardware constraints, or napkin math, the cognitive skills that ML systems interviews demand.

Huyen’s *Designing Machine Learning Systems* (Huyen, 2022) covers ML system design broadly (data engineering, model serving, monitoring) and is widely used for interview preparation. It does not include quantitative practice problems or a structured difficulty progression. Candidates who study the book understand what a feature store is but cannot calculate the memory bandwidth required to serve a 70B model at target throughput.

Table 9 summarizes the positioning. STAFFML addresses the gap between these extremes: the quantitative rigor of LeetCode-style practice (every question has a definite numerical answer) combined with the systems focus of conceptual interview guides (every question is grounded in real hardware constraints). The four-axis classification adds structured difficulty progression and cognitive-mode coverage that no existing platform provides.

12.2 LLM Engineering Benchmarks

Recent benchmarks like HumanEval (Chen et al., 2021) and SWE-bench (Jimenez et al., 2024) evaluate the coding and bug-fixing capabilities of LLM

Table 10. Source material mapping. Each seminal system provides the quantitative content for questions in specific zones. The mapping illustrates how STAFFML operationalizes research into assessment.

System	Concept Tested	Primary Zones	Example Task
Megatron-LM GPipe	Tensor parallelism Pipeline bubbles	Realization, Fluency Realization, Optimization	Size the TP degree for 70B on $8 \times H100$ Calculate bubble fraction for k micro-batches
ZeRO / DeepSpeed	Memory partitioning	Fluency, Diagnosis	Budget optimizer state across N GPUs
FlashAttention	IO-aware attention	Optimization, Quantify	Derive memory reads: standard vs. tiled
vLLM	KV-cache paging	Diagnosis, Fluency	Calculate fragmentation at 128K context
Orca	Continuous batching	Mastery, Evaluation	Diagnose prefill starvation under load
MLPerf	System benchmarking	Quantify, Analyze	Interpret throughput vs. latency tradeoff

agents. These benchmarks focus on algorithmic generation and repository-level software engineering; they do not assess whether an agent understands the physical deployment constraints of the code it writes. STAFFML fills this gap, serving not only as an interview bank for human engineers but as a rigorous benchmark for evaluating the mechanical sympathy of LLM agents.

STAFFML also offers a structural defense against data contamination, a persistent problem in LLM evaluation where models memorize static benchmarks from their training data. Because STAFFML questions are generated programmatically using hardware constants (Section 7), an evaluation suite can randomize these constants (e.g., changing an A100’s bandwidth from 2.0 TB/s to an arbitrary 2.4 TB/s) to ensure the LLM agent is actually performing the physical reasoning and napkin math, rather than retrieving a memorized answer.

12.3 ML Systems Research

STAFFML questions are grounded in seminal ML systems research. Megatron-LM (Shoeybi et al., 2019) introduced tensor parallelism for training large language models, providing the foundation for questions about parallelism strategies and communication overhead. GPipe (Huang et al., 2019) established pipeline parallelism and the bubble fraction analysis that appears in realization-zone questions. ZeRO (Rajbhandari et al., 2020) and DeepSpeed (Rasley et al., 2020) eliminated memory redundancy in data parallelism, motivating questions about optimizer state partitioning and memory budgeting.

FlashAttention (Dao et al., 2022) demonstrated IO-aware algorithm design, reducing memory reads/writes from quadratic to linear in sequence length, and provides the canonical example of how understanding the memory hierarchy yields algorithmic improvements invisible to complexity

analysis alone. vLLM (Kwon et al., 2023) introduced PagedAttention for KV-cache management, directly inspiring our fluency and diagnosis questions about memory fragmentation and serving throughput. Orca (Yu et al., 2022) pioneered continuous batching (iteration-level scheduling), which motivates our mastery-zone questions about prefill starvation and scheduling tradeoffs.

MLPerf (Mattson et al., 2020; Reddi et al., 2020) established standardized benchmarks for measuring ML system performance: training time, inference latency, and throughput across hardware platforms. The distinction matters: MLPerf measures how well systems perform; STAFFML measures how well humans reason about systems performance. MLPerf’s measurement methodology (standardized workloads, reproducible configurations, hardware-specific baselines) informs the quantitative grounding of STAFFML questions, and several questions directly reference MLPerf benchmarking scenarios. The assessment targets differ: MLPerf evaluates hardware and software implementations; STAFFML evaluates human engineering judgment about those implementations.

These systems are not merely cited; they are the source material for STAFFML questions (Table 10). The questions test whether candidates understand the systems principles these papers discovered.

12.4 Educational Measurement

The assessment design methodology draws on three traditions. Bloom’s revised taxonomy (Anderson et al., 2001; Bloom et al., 1956) provides the six-level cognitive framework that maps to our mastery levels, validated by decades of educational research as a reliable predictor of assessment difficulty (Webb, 1997; Biggs and Collis, 1982). Evidence-Centered Design (Mislevy et al., 2003) provides the three-component assessment architecture (claim, evi-

dence, task features) that structures our construction pipeline. Automatic Item Generation (Gierl and Haladyna, 2013) provides the theoretical foundation for template-based question generation, extended here with hardware-grounded construction and multi-model verification.

The topic taxonomy follows SKOS (W3C, 2009) for knowledge organization, faceted classification principles (Hjørland, 2013) for orthogonal axis design, and test blueprinting methodology (Millman and Greene, 1989; Case and Swanson, 2002) for coverage validation. Item Response Theory (Lord, 1980; Hambleton et al., 1991) provides the psychometric framework for future difficulty calibration once sufficient response data (≥ 30 responses per item) is collected.

The TinyTorch educational framework (Reddi, 2026d) provides a complementary pedagogical approach: where STAFFML assesses whether engineers have ML systems competencies, TinyTorch teaches those competencies through a build-from-scratch curriculum spanning 20 modules.

13 Limitations

No empirical validation. The bank is checked for structural coherence, physics applicability, and mathematical consistency, but it has not yet been psychometrically validated against candidate response data. Correlating zone-level performance with engineering seniority is an important next step. Until such data exist, zone and level assignments reflect the design rubric rather than measured item statistics, and the contribution claim is the assessment apparatus, not its calibration.

Human review is staged, not universal. The corpus distinguishes automated validation and math verification from human review. Automated checks catch many structural, arithmetic, and consistency errors, but they do not imply that every item has been independently reviewed by a domain expert. We therefore treat the bank as a living assessment resource: generated or contributed items enter as drafts, pass machine validation, and are promoted only after review proportional to their risk and visibility.

Static questions with chain-based progression. While individual questions are static text, the depth chain structure enables adaptive progression: a candidate who answers an L3 question correctly is presented with the L4 question on the same topic. This simulates some of the adaptivity of live interviews, though it does not replicate the free-form dialogue

of whiteboard sessions. The chains (Section 3.5) approximate the probing behavior of experienced interviewers.

Architectural paradigm drift. The topic taxonomy is anchored in the current Transformer-dominant era, and identifiers such as `attention-mechanism-systems` will need revision as alternative architectures (e.g., state space models (Gu and Dao, 2023)) mature. The four-skill competency model and the five laws of Section 1 are stable under such transitions: any architecture still has an arithmetic intensity, a memory hierarchy footprint, and a communication tax, so the constants shift but the cognitive acts (diagnosing a memory wall, specifying a latency bound, evaluating a parallelism strategy) do not. As new architectures stabilize, topic labels will be deprecated and replaced while questions targeting the same zones are re-grounded in the new constants.

Hardware vendor concentration. The corpus draws predominantly on NVIDIA hardware specifications (H100, A100, Orin), reflecting current industry adoption. Candidates with primary experience on AMD, Intel, or Google TPU hardware may find the specific constants unfamiliar, though the reasoning patterns (roofline analysis, memory budgeting, power-performance tradeoffs) are vendor-neutral. Future versions will include multi-vendor question variants. For assessment contexts, we recommend providing a hardware specification reference sheet alongside the questions, ensuring that recall of vendor-specific numbers does not gate access to the reasoning being tested.

Hardware constant maintenance. Every question is grounded in specific hardware specifications (H100: 80 GB HBM3, 3.35 TB/s, 494 TFLOPS TF32 dense; Orin: 275 TOPS INT8). These numbers are correct as of the current accelerator generation but require updates as new hardware ships. The shared infrastructure modeling framework centralizes constants to make updates tractable. Changing one constant propagates to all affected questions, and the invariant system flags any question whose napkin math references outdated specifications. The methodology is designed to be timeless even as specific numbers evolve.

Applicability matrix lags taxonomy. The applicability matrix that excludes 83 topic-track pairs was originally curated against the 79-topic taxonomy used in the v0 release. The taxonomy has since grown to 87 curated topics with eight v1 additions covering

autograd and computational graphs, chiplet architecture, communication–computation overlap, disaggregated serving, model adaptation systems, recommendation systems engineering, software portability, and sustainability and carbon accounting. Per-track applicability for these eight topics inherits from their nearest pre-v1 analogues until each cell is individually adjudicated; the next taxonomy revision will close this gap with explicit physics justifications for each new topic–track pair.

14 Conclusion and Future Work

STAFFML introduces a structured approach to assessing ML systems engineering expertise. The contribution is the *assessment apparatus*: a 9,521-question corpus, a four-skill competency model whose intersections yield eleven qualitatively different cognitive zones, items that require napkin math against real hardware specifications, and an invariant suite that rejects inconsistent taxonomy, physics, and cognitive tags at commit time. Together these give interviewers, candidates, and educators a shared language for what “knowing” ML systems means across seniority. Empirical IRT-style calibration and large-scale response validation are the natural next step; the present release establishes the schema, content, and quality pipeline that such studies would build on.

The four-axis classification operationalizes this model into an assessment system. The topic taxonomy organizes 87 concepts with prerequisite relationships that enable personalized study paths. The track axis grounds every question in a specific physics regime, ensuring assessment reflects real deployment constraints across cloud, edge, mobile, and TinyML. The level axis maps to both Bloom’s cognitive taxonomy and industry engineering ladders, making the resource actionable for interview preparation and role calibration.

We plan to implement Item Response Theory (IRT) calibration once responses are collected, assigning empirical difficulty and discrimination parameters to each item and measuring the point-biserial correlation between candidate ladder levels and compound-zone performance as a construct validity check. To expand the corpus, a contribution pipeline with automated schema validation will enable industry practitioners to submit questions grounded in real production failure modes. We also envision adaptive question selection that adjusts to individual profiles, targeting the zones where a candidate is weakest.

Incomplete-information and visual-reasoning items are being introduced as draft formats: they provide partial data, diagrams, or operational traces and require candidates to identify what additional information they need before solving the problem. The prerequisite graph will also enable graph-guided study paths that sequence topics based on dependency structure rather than arbitrary ordering.

In the near term, expansion will remain coverage-driven and review-gated: draft questions are used to strengthen weak cells in the empirical coverage tensor, but publication depends on math checks, topic–track validity, and human or community review signals. This keeps the bank useful as a practice resource while making clear that formal calibration and large-scale human validation are ongoing work.

The corpus, schema, and quality infrastructure are available as a companion to the *Machine Learning Systems* textbook (Reddi, 2026b)³.

References

- Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gula-vani, Alexey Tumanov, and Ramachandran Ram-jee. Efficient llm inference via chunked prefills. *ACM SIGOPS Operating Systems Review*, 59(1):9–16, 2024. ISSN 0163-5980. doi: 10.1145/3759441.3759444. URL <https://doi.org/10.1145/3759441.3759444>. Sarathi-Serve: chunked prefill with priority scheduling to bound TTFT under continuous batching.
- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, volume 2023, pages 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.298. URL <https://doi.org/10.18653/v1/2023.emnlp-main.298>. Grouped-query attention: KV-head reduction reduces KV-cache memory at modest quality cost.
- AMD. AMD Instinct MI300X accelerator datasheet. Technical report, Advanced Micro Devices, 2023. URL <https://www.amd.com/en/products/accelerators/instinct/mi300/mi300x>.

³mlsysbook.ai/staffml

- [html](#). MI300X: 192 GB HBM3, 5.3 TB/s memory bandwidth, 1.3 PFLOPS FP16 dense.
- Lorin W. Anderson, David R. Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Pintrich, James Rath, and Merlin C. Wittrock. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman, 2001. The revised Bloom's taxonomy: Remember, Understand, Apply, Analyze, Evaluate, Create.
- AnKing Team. AnKing step deck — community quality control, 2024. URL <https://www.ankingmed.com>. Large-scale Anki deck with community-driven dedup and quality flags.
- Arm Limited. Arm Cortex-M4 technical reference manual. Technical report, Arm Limited, 2020. URL <https://developer.arm.com/documentation/100166/0001>. Cortex-M4: 64 MHz typical, 256 KB SRAM, no MMU, FPU optional, hard real-time.
- John B. Biggs and Kevin F. Collis. *Evaluating the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, 1982. SOLO taxonomy: prestructural, unistructural, multistructural, relational, extended abstract.
- Benjamin S. Bloom, Max D. Engelhart, Edward J. Furst, Walker H. Hill, and David R. Krathwohl. *Taxonomy of Educational Objectives: The Classification of Educational Goals*. David McKay Company, 1956. Original Bloom's taxonomy. Revised by Anderson & Krathwohl (2001).
- Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, pages 21–29. IEEE Comput. Soc, 1997. doi: 10.1109/sequen.1997.666900. URL <https://doi.org/10.1109/sequen.1997.666900>. MinHash for near-duplicate detection at web scale.
- Susan M. Case and David B. Swanson. *Constructing Written Test Questions for the Basic and Clinical Sciences*. National Board of Medical Examiners, 3rd edition, 2002. USMLE test blueprinting with 4+ independent classification axes per item.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. HumanEval: coding benchmark analogous to StaffML's napkin-math assessment.
- Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems* 35, volume 35, pages 16344–16359. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.52202/068431-1189. URL <https://doi.org/10.52202/068431-1189>. IO-aware attention algorithm that reduces memory reads/writes from quadratic to linear in sequence length.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2023. URL <https://arxiv.org/abs/2210.17323>. GPTQ: layer-by-layer one-shot post-training INT4 quantization for LLMs.
- Mark J. Gierl and Thomas M. Haladyna. *Automatic Item Generation*. Routledge, 2013. ISBN 9781136636899. doi: 10.4324/9780203803912. URL <https://doi.org/10.4324/9780203803912>. Foundational text on template-based assessment item generation.
- Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. URL <https://arxiv.org/abs/2312.00752>. Selective state-space models: linear-time scaling, alternative to attention for long sequences.
- Ronald K. Hambleton, Hariharan Swaminathan, and H. Jane Rogers. *Fundamentals of Item Response Theory*. SAGE Publications, 1991. ISBN 9780803936478. Practical guide to IRT. 30+ responses needed for stable calibration.
- Birger Hjørland. Facet analysis: The logical approach to knowledge organization. *Information Processing & Management*, 49(2):545–557, 2013. ISSN 0306-4573. doi: 10.1016/j.ipm.2012.10.001. URL <https://doi.org/10.1016/j.ipm.2012.10.001>. Faceted classification: independent orthogonal axes rather than a single hierarchy.

- Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, pages 103–112. Curran Associates Inc., 2019. URL <https://papers.nips.cc/paper/8305-gpipe-efficient-training-of-giant-neural-networks-using-pipeline-parallelism>. Pipeline parallelism for training models that exceed single-device memory.
- Chip Huyen. *Designing Machine Learning Systems*. O’Reilly Media, 2022. ISBN 978-1098107963. ML systems design textbook focused on production deployment.
- Vijay Janapa Reddi. CS249r: Tiny machine learning — harvard university, 2024. URL <https://sites.google.com/g.harvard.edu/cs249-tinyml-2024>. Graduate course on ML systems spanning cloud to embedded deployment.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2024.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL <https://doi.org/10.1145/3600006.3613165>. vLLM: virtual memory paging for KV-cache reduces fragmentation and enables higher throughput.
- LeetCode Inc. LeetCode — online coding platform, 2024. URL <https://leetcode.com>. 2,869+ validated coding problems with community-driven difficulty calibration.
- Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. MCUNet: Tiny deep learning on IoT devices. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 11711–11722. Curran Associates Inc., 2020. URL <https://papers.nips.cc/paper/2020/hash/86c51678350f656dcc7f490a43946ee5-Abstract.html>.
- Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *GetMobile: Mobile Computing and Communications*, 28(4):12–17, 2024. ISSN 2375-0529, 2375-0537. doi: 10.1145/3714983.3714987. URL <https://doi.org/10.1145/3714983.3714987>. AWQ: salient-weight-aware INT4 weight-only quantization for LLM serving.
- Frederic M. Lord. *Applications of Item Response Theory to Practical Testing Problems*. Routledge, 1980. ISBN 9781136557248. doi: 10.4324/9780203056615. URL <https://doi.org/10.4324/9780203056615>. Foundational IRT text. b-parameter (difficulty), a-parameter (discrimination).
- Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debojyoti Dutta, Udit Gupta, Kim Hazelwood, Andrew Hock, Xinyuan Huang, Atsushi Ike, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St John, Carole-Jean Wu, Lingjie Xu, Cliff Young, and Matei Zaharia. MLPerf Training Benchmark. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 2, pages 336–349. mlsys.org, 2020. URL <https://arxiv.org/abs/1910.01500>. The original MLPerf Training benchmark establishing standardized ML system measurement.
- Samuel Messick. Validity of psychological assessment: Validation of inferences from persons’ responses and performances as scientific inquiry into score meaning. *Am. Psychol.*, 50(9):741–749, 1995. ISSN 1935-990X, 0003-066X. doi: 10.1037/0003-066x.50.9.741. URL <https://doi.org/10.1037/0003-066x.50.9.741>. Unified validity framework: content, construct, consequential validity.

- Jason Millman and Jennifer Greene. The specification and development of tests of achievement and ability. In Robert L. Linn, editor, *Educational Measurement*, pages 335–366. American Council on Education / Macmillan, 3rd edition, 1989. Test blueprinting and tables of specifications; standard reference for coverage validation.
- Robert J. Mislevy, Russell G. Almond, and Janice F. Lukas. A brief introduction to evidence-centered design. *ETS Research Report Series*, 2003(1), 2003. ISSN 2330-8516, 2330-8516. doi: 10.1002/j.2333-8504.2003.tb01908.x. URL <https://doi.org/10.1002/j.2333-8504.2003.tb01908.x>. ECD framework: claims, evidence, task features for assessment design.
- NeetCode. NeetCode 150 — curated coding interview problems, 2024. URL <https://neetcode.io/practice>. 150 problems organized by pattern with difficulty progression.
- NVIDIA Corporation. NVIDIA H100 tensor core GPU architecture. Technical report, NVIDIA Corporation, 2022a. URL <https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper>. H100 datasheet: 80 GB HBM3, 3.35 TB/s, 989 TFLOPS FP16 dense, 494 TFLOPS TF32 dense.
- NVIDIA Corporation. NVIDIA Jetson AGX Orin series technical brief. Technical report, NVIDIA Corporation, 2022b. URL <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/>. Orin AGX: 275 TOPS INT8 sparse, 60 W power envelope, 32 GB LPDDR5.
- OpenAI. Gpt-4 technical report. Technical report, OpenAI, 2023.
- Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020. doi: 10.1109/sc41405.2020.00024. URL <https://doi.org/10.1109/sc41405.2020.00024>. ZeRO optimizer partitioning eliminates memory redundancy in data parallelism.
- Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506. ACM, 2020. doi: 10.1145/3394486.3406703. URL <https://doi.org/10.1145/3394486.3406703>. Training system combining ZeRO, pipeline parallelism, and mixed precision for 100B+ parameter models.
- Vijay Janapa Reddi. *Machine Learning Systems at Scale*. MIT Press, 2026a. URL <https://mlsysbook.ai>. Volume II of the Machine Learning Systems textbook.
- Vijay Janapa Reddi. *Machine Learning Systems*. MIT Press, 2026b. URL <https://mlsysbook.ai>. Two-volume textbook on ML systems following the Hennessy & Patterson pedagogical model. Volume I covers single-machine systems; Volume II covers distributed systems at scale.
- Vijay Janapa Reddi. MLSys-im: First-principles infrastructure modeling for machine learning systems, 2026c. URL <https://mlsysbook.ai/mlsysim>. Companion analytical modeling framework for the ML Systems textbook. Provides shared hardware constants, roofline analysis, and quantitative verification of systems reasoning.
- Vijay Janapa Reddi. Tinytorch: Building machine learning systems from first principles. *arXiv preprint arXiv:2601.19107*, 2026d. Educational ML framework with 20 modules teaching ML as systems engineering from first principles.
- Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020. doi: 10.1109/isca45697.

- 2020.00045. URL <https://doi.org/10.1109/isca45697.2020.00045>. Standardized benchmarks for ML system performance across inference scenarios.
- Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3980–3990. Association for Computational Linguistics, 2019. doi: 10.18653/v1/d19-1410. URL <https://doi.org/10.18653/v1/d19-1410>. Sentence embeddings used for semantic deduplication of question scenarios.
- Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. Tensor parallelism for training large language models across GPUs.
- Dagobert Soergel. *Organizing Information: Principles of Data Base and Retrieval Systems*. Academic Press, 1985. User warrant vs literary warrant in knowledge organization.
- Yangshun Tay. Grind 75: Interview problem list. Tech Interview Handbook, 2024. URL <https://www.techinterviewhandbook.org/grind75>. 169 problems ranked by frequency, pattern coverage, and difficulty.
- Martin Thompson. Mechanical sympathy, 2011. URL <https://mechanical-sympathy.blogspot.com/>. Blog. The term describes software that works with the hardware rather than against it, inspired by racing driver Jackie Stewart’s philosophy.
- W3C. SKOS simple knowledge organization system reference. W3c recommendation, World Wide Web Consortium, 2009. URL <https://www.w3.org/TR/skos-reference/>. Standard for broader/narrower/related concept relationships.
- Norman L. Webb. Criteria for alignment of expectations and assessments in mathematics and science education. *Research Monograph No. 6*, 6, 1997. Depth of Knowledge (DOK) framework: four levels of cognitive complexity for assessment alignment.
- Grant Wiggins and Jay McGighe. *Understanding by Design*. Association for Supervision and Curriculum Development, 2nd edition, 2005. Backward design methodology: desired results, acceptable evidence, then learning plan.
- Samuel Williams, Andrew Waterman, and David Patterson. Roofline. *Communications of the ACM*, 52 (4):65–76, 2009. ISSN 0001-0782, 1557-7317. doi: 10.1145/1498765.1498785. URL <https://doi.org/10.1145/1498765.1498785>. Canonical roofline model: arithmetic intensity, ridge point, compute- vs memory-bound classification.
- Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 521–538. USENIX Association, 2022. Iteration-level scheduling (continuous batching) for transformer serving, eliminating padding waste.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36*, pages 46595–46623. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023. doi: 10.52202/075280-2020. URL <https://doi.org/10.52202/075280-2020>. NeurIPS 2023 Datasets and Benchmarks track.