

A FIELD GUIDE FOR NETWORK ENGINEERS

Building the AI Factory

Designing and operating the networks behind AI training and inference: GPU fabrics, RoCEv2 and InfiniBand, Ultra Ethernet, scale-up and the front-end.

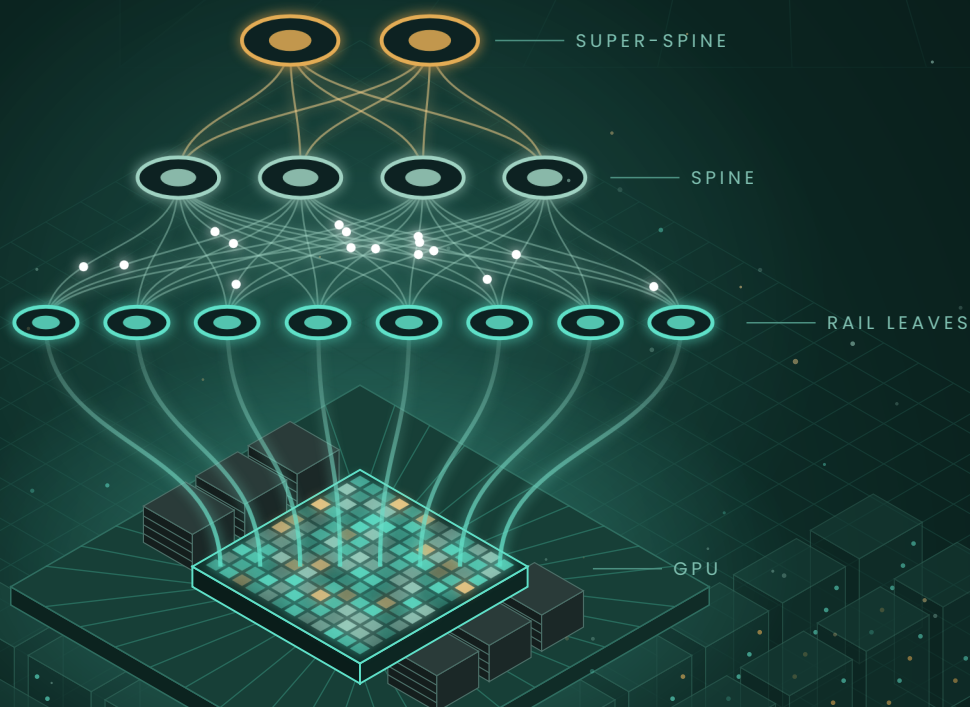
GPU FABRICS

LOSSLESS ROCE

LOAD BALANCING

ULTRA ETHERNET

INFINIBAND



NetworkFactory
networkfactory.net

18 chapters, 3 appendices
Reference platform: NVIDIA Spectrum, Cumulus Linux and SONiC

Building the AI Factory

A field guide for network engineers. Complete edition, 2026.

Copyright © 2026 NetworkFactory. All rights reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and other noncommercial uses permitted by copyright law.

Published by NetworkFactory, networkfactory.net

Disclaimer and limitation of liability

This book is provided for general educational and informational purposes only. Its contents are provided "as is" and "as available", without warranty of any kind, express or implied, including but not limited to warranties of accuracy, completeness, currency, merchantability, fitness for a particular purpose and non-infringement.

To the fullest extent permitted by applicable law, the author, publisher and any contributors shall not be liable to any person or entity for any loss, damage, outage, data loss, security incident, cost or expense of any kind, whether direct, indirect, incidental, special, consequential or punitive, arising out of or in connection with the use of, reliance on, or inability to use any information, design, configuration, calculation or code in this book, even if advised of the possibility of such damages. Readers assume full responsibility for their use of the material.

Configurations, commands, scripts and designs are illustrative reference examples. Software syntax, defaults and capabilities change between releases and platforms. Before applying anything from this book to any network, validate it in a non-production lab, consult the current official documentation of the vendors concerned, and follow your organization's change management, security and safety procedures. Nothing in this book constitutes professional engineering, legal, security, compliance or financial advice.

Performance figures, prices, power values, product specifications and capacity examples are approximate and illustrative, reflect publicly available information at the time of writing, and may not apply to your environment.

Trademarks and independence

NVIDIA, Spectrum, Spectrum-X, Cumulus, Cumulus Linux, NVUE, ConnectX, BlueField, Quantum, NVLink, NVSwitch, GPUDirect, NCCL, UFM, DGX and SuperPOD are trademarks or registered trademarks of NVIDIA Corporation. InfiniBand is a trademark of the InfiniBand Trade Association. Cisco and CCDE are trademarks or registered trademarks of Cisco Systems, Inc. Linux is a registered trademark of Linus Torvalds. All other product, company and brand names are the property of their respective owners and are used for identification purposes only.

This book is an independent publication. It is not affiliated with, sponsored, authorized or endorsed by NVIDIA Corporation, Cisco Systems, Inc., the InfiniBand Trade Association, the Ultra Ethernet Consortium or any other vendor or organization mentioned.

Certification content

References to the CCDE certification, including the mapping in Appendix D, are for study context only and reflect the author's reading of Cisco's public exam documents. The scenarios, questions and checklists in this book are original material written for learning purposes; they are not taken from, and do not represent, actual examination content.

Fictitious organizations

Companies and organizations used in examples and scenarios, including Northstar Labs, Solvane, Meridian Health and others, are fictitious. Any resemblance to real organizations is coincidental.

Preface

Network engineers have spent two decades perfecting the data center: leaf-spine fabrics, BGP everywhere, VXLAN EVPN overlays, automation. Then AI arrived, and many of the instincts that served us well started to mislead us. Oversubscription that was always safe became a costly mistake. Averages that were always good enough began hiding the problems that mattered. A dropped packet, once a non-event, could stall thousands of GPUs.

This book exists to bridge that gap. It is written for engineers who already know how to build networks and now need to build the networks behind AI training and inference: to understand why they are different, how to design them, how to configure them and how to keep them running. It assumes the networking and explains the rest, starting from what a GPU actually does and ending with complete designs.

Three convictions shaped it. First, **understanding why comes before memorizing what**; every technology is introduced with a plain-language picture before its mechanics. Second, design is judgment, and judgment is built by tracing decisions back to requirements; the decision logs in Part V make that explicit. Third, configuration examples should be concrete enough to try, and honest about their limits; every one is flagged with the release it was checked against and the advice to validate it in a lab.

The book grew out of preparation for the CCDE Practical exam and its AI Infrastructure area of expertise. That is why **CCDE lens** boxes appear throughout, why Appendix C condenses the book into a scenario checklist, and why Appendix D maps every chapter to Cisco's public AI Infrastructure technology list. It is not a Cisco book, and not a book for one vendor. **The design principles are vendor-neutral**: the same physics, protocols and trade-offs apply whether your fabric runs NVIDIA, Arista, Cisco, Juniper or SONiC, and the certification simply provided a rigorous syllabus to test them against.

Whether you are preparing for a design certification, moving your career toward AI infrastructure, or responsible for an AI cluster arriving next quarter, we hope this book makes the path shorter and the designs better.

NetworkFactory

Chapter 1

The AI factory

What an AI data center produces, the jobs it runs, and why its network is built differently from every data center you have designed before.

IN THIS CHAPTER YOU WILL

- Explain what an AI factory produces and why parts of it behave as one machine
- Distinguish pre-training, fine-tuning, inference and RAG by their network demands
- Name the four networks and the scale-up domain
- Separate synchronized from independent traffic, the idea behind every later design choice

1.1 Why call it a factory

A traditional data center is closer to an office building. It hosts many independent applications for many independent users: email, ERP, websites, databases. Each application does its own small amount of work, and the network's job is to connect thousands of unrelated conversations reliably.

An AI data center behaves differently. Large parts of it work as **one machine**, producing one product. Thousands of GPUs cooperate on a single training job for days or weeks, and the output is a trained model. Then that model is put to work answering requests, and the output is generated text, images or decisions. NVIDIA and others call this kind of facility an **AI factory**, and the name is useful for a network engineer because a factory makes the network's role obvious: it is the conveyor system that keeps every worker supplied, and when the conveyor stalls, the whole line stops.

ANALOGY

Picture a car factory. The **workers** are the GPUs. The **assembly line** connects workers building the same car together: that is the back-end network. The **warehouse and its trucks** bring raw materials in and store finished parts: the storage network. The **front office and customer counter** handle orders and deliveries: the front-end network. The **maintenance crew's walkie-talkies** reach every machine even when everything else is broken: the out-of-band management network. Every chapter of this book adds detail to one of these five roles.

1.2 What the factory produces

Models

An AI model is, physically, a very large file of numbers called **parameters** or **weights**. During training, these numbers are adjusted, step by step, until the model gets good at its task. A model with 70 billion parameters stored at 16 bits each occupies about 140 GB, which is a useful number to keep in mind: whenever a model is copied, loaded or saved, that much data moves across a network.

Tokens

Language models read and write in **tokens**, small pieces of text roughly the size of a short word or part of a word. When a model answers a question, it generates its response one token at a time. That makes tokens the factory's unit of output: inference platforms are measured in tokens per second, and their customers often pay per token.

1.3 The jobs the factory runs

Almost everything that happens on an AI platform falls into four kinds of work. They demand very different things from the network, and recognizing which one you are designing for is the first step in every design.

Pre-training

Teaching a model from scratch on an enormous dataset. This is the largest job in AI: thousands to tens of thousands of GPUs working on a single job for weeks or months. Every GPU must exchange results with the others at every training step. Pre-training is where the most demanding back-end fabrics in the world are built.

Fine-tuning

Taking an already-trained model and continuing its training on a narrower dataset, for example a bank's internal documents, so it becomes an expert in that domain. It is like sending a general doctor through a specialist residency. **Full fine-tuning** adjusts all of the model's weights and behaves like a smaller version of pre-training. **Parameter-efficient fine-tuning**, such as **LoRA** (low-rank adaptation), freezes the original weights and trains a small set of additional ones instead. It needs far less compute and memory, so it often fits on a single server or a handful of servers, which changes the network requirements dramatically.

Inference

Using a trained model to answer requests. Each request is small, but there are many of them, arriving independently from many users. Inference runs in two phases worth knowing by name:

- **Prefill:** the model reads the whole prompt at once. This is compute-heavy and determines the **time to first token**, how long a user waits before the answer starts appearing.
- **Decode:** the model generates the answer one token after another. This phase is limited mostly by memory bandwidth, and determines the **time per output token**, how fast the answer streams.

During decode, the model keeps a working memory of the conversation so far, called the **KV cache**. When a model is small enough to fit within one server, inference servers work alone and need no back-end fabric at all. When a model is too large for one server, or when the prefill and decode phases are split across different servers to use them more efficiently (called **disaggregated serving**), inference servers must exchange data such as the KV cache, and inference starts to need a fast back-end of its own.

Retrieval-augmented generation (RAG)

Before answering, the model first looks up relevant documents and includes them in its prompt, so it can answer using information it was never trained on. The lookup goes to a **vector database**, which finds documents by meaning rather than by exact words. For the network, RAG adds east-west traffic between inference servers and the database on every request, and where that database sits relative to the inference servers can decide whether the front-end needs more capacity. Models increasingly also call tools and services while answering, for example through the Model Context Protocol (MCP), an open standard for connecting models to data sources and applications. Each such call is more front-end traffic, and a new path from the inference servers to other systems that must be secured (Chapter 14).

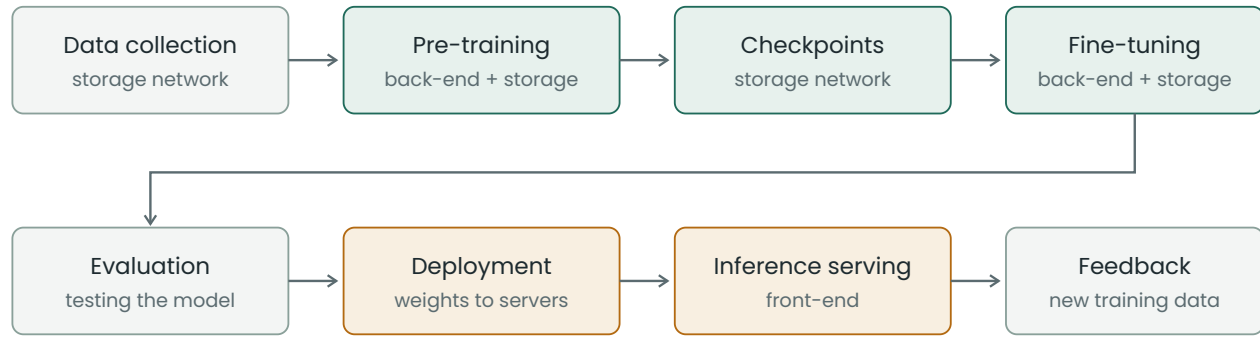
Workload	Scale of one job	Traffic pattern	What the network must protect
Pre-training	Thousands of GPUs, weeks	Synchronized GPU-to-GPU at every step; huge dataset reads; periodic checkpoints	Job completion time; lossless, non-blocking back-end
Full fine-tuning	Tens to hundreds of GPUs, hours to days	Same pattern as pre-training, at smaller scale	Job completion time
Parameter-efficient fine-tuning	One to a few servers	Mostly inside one server; light network use	Storage access; often no large back-end needed
Inference (model fits one server)	One server per request	Independent requests; north-south; weight loads from storage	Response latency at peak; front-end capacity
Inference (large or disaggregated)	Several servers per request	Independent requests plus GPU-to-GPU transfers such as the KV cache	Latency; a back-end for the inference fleet
RAG	Adds to each inference request	East-west lookups to the vector database	Locality between inference and retrieval services

DESIGN PRINCIPLE

Identify the workload before you draw a single switch. The same 1,000 GPUs need a completely different network depending on whether they pre-train one model, fine-tune many small ones, or serve inference.

1.4 The life of a model

Most AI platforms run all of these jobs over the life of a model, often at the same time on different parts of the same facility. Following one model through its life shows where each network does its work.



Feedback becomes new data, and the cycle repeats with the next version of the model.

Figure 1.1 The life of a model, and the network that does the work at each stage.

1. **Data collection.** Raw data (text, images, sensor recordings) lands on the storage arrays. This is mostly a storage-network and ingest problem.
2. **Pre-training.** The back-end carries synchronized GPU traffic for weeks; the storage network feeds data and absorbs checkpoints.
3. **Checkpoints.** Snapshots of the model are written regularly so a failure costs minutes of work rather than weeks.
4. **Fine-tuning.** The model is specialized for a domain or customer, on the same networks at smaller scale.
5. **Evaluation.** The model is tested before release, usually with inference-like jobs.
6. **Deployment.** The final weights are copied to inference servers. Every server that will serve the model must load them, which can create large bursts on the storage network.
7. **Inference serving.** Users send requests through the front-end, and the model answers.
8. **Feedback.** Real-world use generates new data, and the cycle begins again.

1.5 The factory floor plan

Every AI facility, from a 64-GPU lab cluster to a 100,000-GPU campus, is built from the same parts. What changes is their size and which of them are needed.

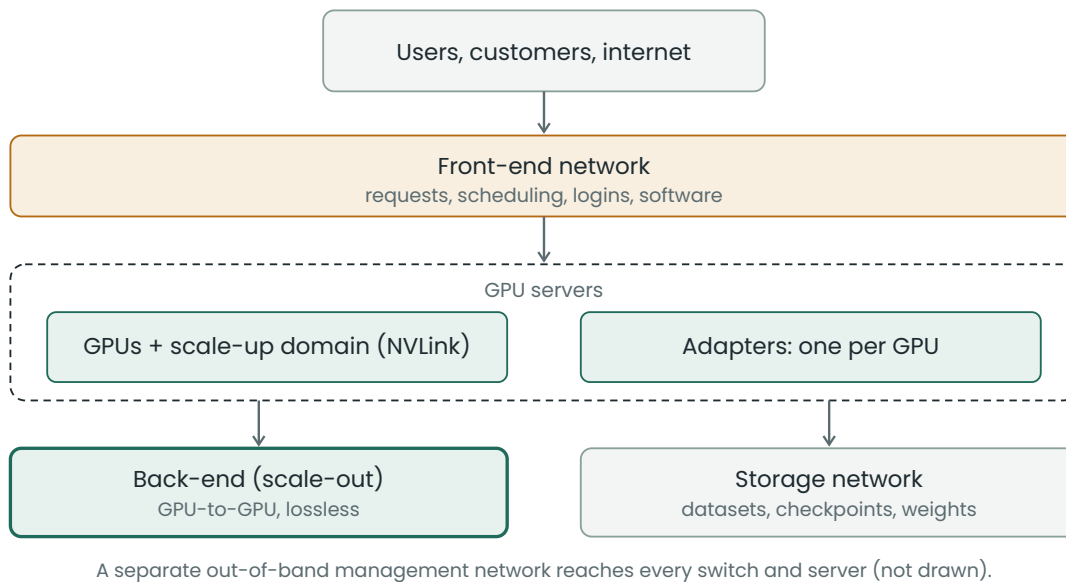


Figure 1.2 The parts of every AI factory.

The scale-up domain

Inside a GPU server, the GPUs are joined by a private, extremely fast interconnect. On NVIDIA systems this is **NVLink**, switched by **NVSwitch** chips. It is roughly an order of magnitude faster than any network port leaving the server. Making this domain bigger is called **scale-up**.

Traditionally it stopped at the edge of one 8-GPU server; rack-scale systems now extend NVLink across a whole rack, joining 72 GPUs in one domain. Chapter 2 opens the server up in detail.

The back-end, or scale-out, network

Connecting many servers so their GPUs can work on one job is called **scale-out**, and the network that does it is the **back-end**. It carries only GPU-to-GPU traffic, it is typically built to be lossless and non-blocking, and it uses **RDMA**, either over **InfiniBand** or over Ethernet as **RoCEv2**. Part III is devoted to it.

The storage network

Connects every server to shared storage: datasets in, checkpoints out, model weights to inference servers. Its defining challenge is convergence: thousands of servers reading from and writing to a comparatively small number of storage nodes, often all at once. Chapter 7 covers it.

The front-end network

Carries everything humans and control systems do: user and customer requests, the job scheduler, monitoring, engineer logins, software downloads. Its traffic is independent and TCP-based, so it is built like a well-designed conventional data center fabric, typically leaf-spine with VXLAN EVPN. Chapter 6 covers it.

The out-of-band management network

A small, separate network connecting the management and console ports of every switch and the management controllers of every server. It must keep working when everything else fails, because that is exactly when engineers need it. Chapter 8 covers it.

DESIGN PRINCIPLE

Networks are defined by the kind of traffic they carry, not by the kind of server attached to them. A training server typically connects to all four networks at once, through different ports, each carrying a different kind of traffic.

1.6 Two kinds of traffic

If you remember one idea from this chapter, make it this one. AI facilities carry two fundamentally different kinds of traffic, and almost every design decision follows from which kind you are dealing with.

Synchronized traffic: back-end during training



Independent traffic: front-end requests



Time runs left to right. Shaded blocks are moments when each sender transmits.

Figure 1.3 Synchronized traffic peaks together; independent traffic spreads out.

Synchronized traffic

During training, every GPU computes on its share of the data, then all GPUs exchange their results before anyone can continue. They all transmit at the same moment, then fall silent while they compute, then transmit together again. Chapter 3 explains the mechanism, called a **collective** such as **all-reduce**.

ANALOGY

A rowing crew. The boat moves at the speed of the slowest rower, and every rower pulls at the same instant. If one rower is delayed by even a moment, the whole crew waits. In a training cluster, if one link is congested during an exchange, every other GPU in the job sits idle until the slowest transfer finishes.

Two consequences follow, and they shape the entire back-end:

- **The worst case is the normal case.** Because every GPU transmits at once, the network must carry full traffic from all of them simultaneously. Sharing capacity on the bet that not everyone sends at once (oversubscription) is a losing bet here.
- **Tail latency rules.** What matters is not the average transfer time but the slowest one, because everyone waits for it. A network that is fast on average but occasionally slow is a slow network for training. The arithmetic is unforgiving: if each transfer is slow only once in a million times, a training step made of a million transfers contains at least one slow one about 63% of the time $(1 - (1 - 10^{-6})^{1,000,000})$.

Independent traffic

On the front-end, thousands of users and systems send requests that have nothing to do with each other. Their peaks rarely line up, so links can be shared safely, a dropped packet is simply resent by TCP, and one slow request inconveniences one user rather than stalling thousands of GPUs. This traffic can be served by conventional, well-understood network designs.

DESIGN PRINCIPLE

Synchronized traffic gets dedicated, non-blocking, loss-free capacity. Independent traffic gets capacity sized from measured peaks, failure cases and headroom. Treating one like the other is the most common and most expensive mistake in AI network design.

CCDE LENS

A classic scenario trap: a finance stakeholder quotes the back-end's average utilization (say, 45%) and proposes oversubscribing it to save money. The average blends quiet compute phases with moments when every port runs at line rate together. The correct response keeps the training back-end non-blocking, and finds savings where traffic is independent or where a network is not needed at all.

1.7 Why the network decides the economics

GPUs are the most expensive component in an AI facility, and they are only productive while they compute. Every moment a GPU waits for data, from the back-end, from storage or from anywhere else, is paid for and wasted. The network's job, above everything else, is to **keep the GPUs fed**.

A simple calculation shows the scale. The price used here is an illustrative round number, not a quote.

```
Cluster: 1,024 GPUs running one training job for 30 days
GPU-hours: 1,024 x 24 x 30 = 737,280
If network stalls stretch the job by 20%: 737,280 x 0.20 = 147,456 GPU-hours wasted
At an illustrative $2.50 per GPU-hour: about $369,000 for one job
```

That is why AI networks are judged by **job completion time** for training and by **latency at peak** for inference, rather than by link utilization. A network that looks lightly used but adds a few percent to every training step can cost more than the network itself. It is also why, as later chapters show, the right place to save money is rarely the back-end.

1.8 The questions every design starts with

Before choosing a topology, a technology or a single switch, these are the questions to answer. Each answer has a direct network consequence that later chapters develop.

Question	Why it matters to the network	Chapter
Which workloads: pre-training, fine-tuning, inference, RAG?	Decides whether a back-end is needed, and for which servers	1, 3
What is the largest single job, in GPUs?	Sets the size of one non-blocking fabric; total GPU count does not	5
How many GPUs and network adapters per server, and at what speed?	Sets port counts, number of rails, leaf and spine counts	2, 5
Does the model fit in one server or one scale-up domain?	Decides whether inference needs a back-end	2, 17
InfiniBand or Ethernet for the back-end?	Driven by team skills, tooling, scale and lock-in tolerance	10, 12
Dataset size, read rate and checkpoint behavior?	Sizes the storage network, especially toward the storage nodes	7
Peak request rate and latency targets?	Sizes the front-end from measured peak, failure case and headroom	4, 6
Power and cooling per rack?	Decides density, cable lengths, optics and how far the fabric spans	5, 16
Tenants, compliance and data sovereignty?	Drives isolation design on every network and where data may be processed	14
Growth plan for the next one to two years?	Decides whether today's topology can grow without being rebuilt	5

LAB NOTE

Pick an AI platform you know of, or the one your organization plans. Answer the ten questions above for it in one sentence each, and mark the ones you cannot answer. Those gaps are exactly the questions to ask the business before any design work begins, and the same habit is tested directly in the CCDE Practical, where "what information is missing" is a recurring question type.

Chapter summary

- An AI factory produces trained models and generated tokens. Large parts of it behave as one machine.
- Four kinds of work, pre-training, fine-tuning, inference and RAG, place very different demands on the network. Identify the workload first.
- Every facility is built from the same parts: the scale-up domain inside servers, and four networks outside them (back-end, storage, front-end and out-of-band management).

- Synchronized traffic, which defines the training back-end, needs dedicated, non-blocking, loss-free capacity. Independent traffic can share capacity sized from measured peaks.
- Idle GPUs are the dominant cost. The network is judged by how well it keeps them fed, measured as job completion time and latency at peak.

Key terms

AI factory: a facility whose product is trained models and generated output.

Parameters (weights): the numbers that make up a model.

Token: a small unit of text a language model reads and writes.

Pre-training: training a model from scratch on a very large dataset.

Fine-tuning: further training an existing model on a narrower dataset.

LoRA: a parameter-efficient fine-tuning method that trains a small set of added weights.

Inference: using a trained model to answer requests.

Prefill and decode: the prompt-reading and answer-generating phases of inference.

KV cache: the model's working memory of a conversation during decode.

Disaggregated serving: running prefill and decode on different servers.

RAG: retrieving relevant documents before generating an answer.

Vector database: a database that finds information by meaning.

Scale-up: making one GPU domain larger, for example with NVLink.

Scale-out: connecting many servers over a network.

Back-end network: the GPU-to-GPU scale-out fabric.

Checkpoint: a saved snapshot of training progress.

Collective / all-reduce: an operation in which all GPUs exchange and combine results.

Job completion time: how long a training job takes from start to finish.

Tail latency: the delay experienced by the slowest transfers, rather than the average.

Review questions

1. A company plans 512 GPUs for LoRA fine-tuning of small models, each job fitting on a single server. Should the design start from a large non-blocking back-end? Why or why not?
2. Why is the average utilization of a training back-end a misleading number for capacity planning?
3. An inference model is too large to fit in one server. What changes for the network compared with a model that fits?
4. Name the four networks of an AI factory and the kind of traffic each carries.

5. Why does the largest single job, rather than the total GPU count, determine the size of one back-end fabric?
6. Using the calculation in section 1.7, estimate the GPU-hours wasted if network stalls add 5% to a 14-day job on 2,048 GPUs.

Answers

1. No. If each job fits in one server, GPUs never need to exchange data across servers during training, so a large lossless, non-blocking back-end would carry almost nothing. The design should focus on storage access and the front-end, and size any back-end only for jobs that genuinely span servers.
2. Training traffic is synchronized: quiet compute phases alternate with moments when every port runs at full speed together. The average blends the two and hides the peaks that determine how long each step takes.
3. Serving one request now involves several servers that must exchange data such as model activations or the KV cache. Inference traffic then needs fast, low-latency server-to-server connectivity, effectively a back-end for the inference fleet, in addition to the front-end and storage networks.
4. Back-end: synchronized GPU-to-GPU traffic. Storage: datasets, checkpoints and model weights. Front-end: user requests, scheduling, monitoring, logins and software downloads. Out-of-band management: device management and console access.
5. A job's GPUs must all exchange data with each other, so they must sit on one fabric. Several smaller jobs can run on separate fabrics, but one large job cannot be split across them without crossing a bottleneck between fabrics.
6. $2,048 \times 24 \times 14 = 688,128$ GPU-hours. At 5%, about 34,406 GPU-hours are wasted.

Appendix C

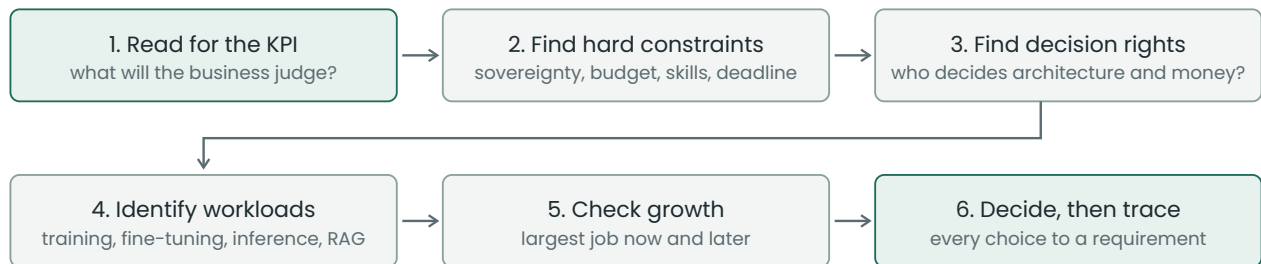
CCDE Practical decision checklist

The whole book condensed into the checks to run on every AI infrastructure scenario, in the order the exam rewards.

THE THREE RULES THAT DECIDE MOST QUESTIONS

1. Anchor on the stated KPI and hard constraints before considering convenience or cost.
2. Ask what the traffic actually does: synchronized or independent, spreading or converging, frequent or rare.
3. Size for the failure day, not the healthy day, and trace every decision to a requirement.

C.1 Reading the scenario



Convenience is only a tiebreaker, after the KPI and the hard constraints.

Figure C.1 Reading a design scenario before answering any question.

- ☐ What metric will the business judge the design on: job completion time, latency, cost, time to first model?
- ☐ Which constraints are hard: data sovereignty, regulation, budget envelope, deadline, hiring freeze, vendor policy?
- ☐ Which document gives decision rights when stakeholders disagree?
- ☐ What is the largest single job today and after growth?
- ☐ What information is missing that would change the design? Ask for it before designing.
- ☐ Which details are noise? Not every fact in a scenario matters.

C.2 Workloads and placement

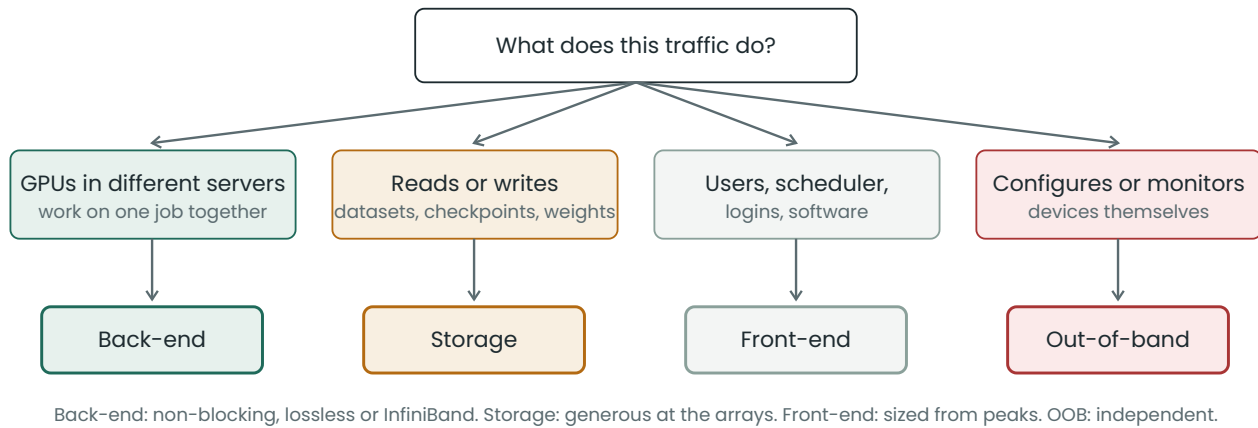


Figure C.2 Which network does this traffic belong on?

- ☐ Do these servers cooperate across servers? If not, no back-end is needed.
- ☐ Does the model fit in one server or one scale-up domain?
- ☐ Is the chattiest parallelism (tensor) inside NVLink?
- ☐ Must data stay where it is collected? Train where data must live; infer where users are.
- ☐ Could locality remove traffic before bandwidth is added?

C.3 Back-end

- ☐ InfiniBand or Ethernet: decided by team skills, tooling, deadline, scale and lock-in tolerance.
- ☐ Non-blocking? Synchronized traffic gets 1:1; averages never justify oversubscription.
- ☐ Fits two tiers? $R \times R / 2$ against the largest job, now and after growth.
- ☐ Rail-optimized with scalable units?
- ☐ Dedicated, not shared with storage bursts or enterprise traffic?
- ☐ Few large flows: entropy, adaptive routing with capable adapters, topology-aware placement.
- ☐ Resilient hashing, so a link failure moves only its own flows.
- ☐ Spines share one ASN (and super-spines another); partial failures handled by weighted ECMP or withdrawal.
- ☐ Scale-up domain size known; tensor and expert parallelism kept inside it; jobs aligned to it.

C.4 Lossless and congestion

- ☐ ECN warns, DCQCN reacts, PFC is the last resort; ECN thresholds below PFC.
- ☐ One DSCP and priority mapping end to end, hosts included.
- ☐ Storage in its own class or network, never sharing the training lossless class.
- ☐ Periodic slowdowns: match the timing to checkpoints.
- ☐ Locate the congestion point first: convergence (last hop, inside a leaf) needs rate control; spreading (leaf, spine, super-spine links) needs better path choice.
- ☐ Spine-to-leaf hot spots with balanced leaf uplinks: spraying, downstream-aware path choice or placement.

C.5 Storage

- ☐ Size toward the convergence point at the storage nodes.
- ☐ Can bursts be hidden (asynchronous), spread (sharded) or removed (cached)?
- ☐ RDMA storage on Ethernet needs a lossless class; TCP storage does not.

C.6 Front-end and resilience

- ☐ Ratio derived from peak, headroom and failure case, not chosen.
- ☐ With N spines, own N minus one; dual-homing makes the partner-leaf failure a second case to size.
- ☐ What does each device's failure take with it? Single-homed leaves are large blast radii.
- ☐ Ratios multiply along the path to the border.

C.7 Tenancy and security

- ☐ Native isolation per layer: VRFs, InfiniBand partitions, storage namespaces and keys.
- ☐ May tenants share a server? Node-exclusive allocation, MIG, cleanup, DPUs.
- ☐ Cross-tenant traffic only through an inspected firewall.
- ☐ Management never reachable by tenants.

C.8 Operations

- ☐ Management out of band; control in band but protected; data in band.
- ☐ Streaming, sub-second telemetry of queues, ECN, pauses and drops; synchronized clocks.
- ☐ Acceptance stages and a recorded baseline.
- ☐ Source of truth, generated configuration, validation, staged rollout, verification.

C.9 Classic traps

Trap	What the scenario says	What to do
Convenience bias	"Reuse the existing fabric" or "deploy now in the racks we have"	Check the KPI and growth first
Averages	"The network is only 45% used"	Synchronized traffic peaks together
Healthy-day sizing	Capacity fits with everything up	Recalculate with one failure
Wrong attachment	Every server on the back-end	Only cooperating servers need it
Convergence blindness	Same ratio everywhere	Be generous where traffic converges
Bandwidth reflex	"Add spines" or "add uplinks"	Try locality and caching first
Unit errors	TB/s next to Gb/s	Multiply bytes by 8
Wrong isolation tool	VLANs on InfiniBand	Partitions
Physics	Low latency across oceans	Move the model, not the request
Spines for incast	"Add spine capacity" for a last-hop or storage bottleneck	Rate control and fan-in; spines do not help convergence
Valley paths	Traffic between two leaves crosses a third after a failure	Shared spine ASN
Rolling the dice again	One link failure slows every job	Resilient hashing

C.10 Working the 2-hour module

- ☐ Spend the first minutes reading every document before the first question; note the KPI, constraints and decision rights.
- ☐ For each question, find the document and sentence that decides it before choosing.
- ☐ Remember there is no going back: when a new document arrives, re-check earlier assumptions for the questions still ahead.
- ☐ When two answers seem right, prefer the one that satisfies more stated requirements, then the simpler one.
- ☐ Do not invent requirements the scenario does not state.

Appendix D

CCDE AI Infrastructure topic map

Where each topic of the CCDE Practical AI Infrastructure technology list is covered in this book, and where you will need other sources.

Cisco publishes a technology list for the AI Infrastructure area of expertise of the CCDE Practical exam (v3.1). It is added to the CCDE core technology list, which this book does not attempt to cover. The table follows the public list's numbering, with topics paraphrased; check Cisco's current document before you rely on it. The coverage column is honest: **Covered** means the book treats the topic at design depth, **Partly** means it appears but you should study it further, and the note says what is missing.

NOT AN OFFICIAL MAPPING

This map is the author's reading of a public Cisco document. It is not endorsed by Cisco and does not reflect any exam content.

Technology list topic	In this book	Coverage
1.0 AI/machine learning, compliance and governance		
1.1 Infrastructure impact of training, fine-tuning and inference (GenAI, ML, deep learning, LLM, SLM)	1.3, 1.4, Ch. 3, 4.5, 17.1–17.5, 18.2	Partly Small language models are not treated separately.
1.2 Service placement: on-premises, cloud or distributed	1.8, 18.2, 18.3	Partly Cloud placement is discussed only briefly.
1.3 Data sovereignty	1.8, 14.7, 18.2	Covered
1.4 Compliance, policies and governance	1.8, 14.5–14.7, 15.2	Partly Governance frameworks are out of scope.
1.5 Moving workloads between sites	18.4, 18.5	Partly Models and inference only; training migration is not covered.
1.6 Sustainability: green AI, cost, accelerator efficiency, power and cooling	1.7, 2.9, 4.4–4.6, 5.5, 5.8	Partly Power, cooling and cost are covered; carbon-aware placement is not.
1.7 Interoperability, multi-cloud and vendor lock-in	1.8, 11.11, 12.10	Partly
2.0 Network		
2.1 Lossless fabrics	Ch. 10, 12.4	Covered
2.2 QoS in lossless fabrics	6.12, 10.3–10.7, 10.9	Covered

2.3 Redundancy, resiliency and disaster recovery	4.6, 6.8, 13.6, 16.3, 17.4	Partly Disaster recovery across sites is not covered.
2.4 Transport: Ethernet, InfiniBand, latency, bandwidth, optics and cables	Ch. 4, 5.8, Ch. 9–12	Covered
2.4.e Single-site versus multi-site	18.3–18.6	Partly
2.5 Traditional and SDN infrastructure, and integration between fabrics	Ch. 6, 12.3, Ch. 15	Partly SDN controllers are not covered.
2.6 Layer 2 scalability and reachability learning	6.5–6.7, 12.2, 12.3	Covered
2.7 Use-case specific routing optimizations	6.3, 6.4, Ch. 11	Covered
2.8 RDMA and RoCE/RoCEv2	Ch. 9, Ch. 10	Covered
2.9 Connectivity models: purpose-built islands or the main data center (Clos, two or three tier)	1.5, Ch. 5	Covered
3.0 Security		
3.1 Hosting implications of AI/ML infrastructure	14.1, 14.2	Covered
3.2 Infrastructure protection: air-gapped, public, private	8.2, 14.5	Partly Air-gapped designs are not covered.
3.3 Protection against malicious use (for example web application firewalls)	6.9, 17.4	Partly Application-layer protection is out of scope.
3.4 Resource security: on-premises, cloud, clusters, containers	14.3, 14.4, 14.6	Partly Containers and Kubernetes are not covered.
4.0 Hardware and environment		
4.1 Compute resources	Ch. 2	Covered
4.2 AI-enabling hardware: GPU, SmartNIC, DPU	2.2, 2.5, 2.7	Covered
4.3 Real-time operations: data scrubbing, log correlation, telemetry analysis	Ch. 13	Partly Telemetry is covered; scrubbing and log correlation are not.
4.4 Storage: strategy, FC, FCoE, NVMe, NVMe-oF, software-defined and IP storage	Ch. 7	Covered
4.5 Timing protocols such as PTP	8.3, 8.5	Partly Mentioned; not designed in depth.

D.1 Filling the gaps

The partly covered topics share a pattern: they sit at the edge of the network, where AI infrastructure meets the organization, the facility and the application. A study plan for them:

- **Governance, compliance and sovereignty:** map each requirement to a placement decision (where data may live, where models may run), the same way Chapter 18 does for the edge.
- **Multi-site and disaster recovery:** decide what is replicated (checkpoints, model weights, datasets) and what is rebuilt, and size the interconnect for the checkpoint, not the average.
- **Sustainability:** treat power per rack and cooling capacity as hard constraints, and GPU utilization as the efficiency metric that the network protects.
- **Security beyond the fabric:** air-gapped designs, container platforms and application-layer protection for inference endpoints.
- **Timing:** where PTP is required, which devices act as boundary clocks, and how the management plane carries it.

For every topic, practice the habit the exam rewards: state the requirement, name two options, and record which requirement decided between them.