The Foundation Model Paradigm Shift

Traditional AI
(Task-Specific Modeling)

Train from Scratch
(Specialized Data & Compute)

Foundation Models (FMs)

LLMs (Text) | LMMs (Multimodal)

General-Purpose Adaptation
- Prompt Engineering & Fine-tuning
- Accelerated Time-to-Market

Training Scale Tradeoff

Massive Scaling
(Data, Parameters, Compute)

Improved Capabilities

Tradeoff: Limited Access necessitates reliance on Model-as-a-Service APIs vs. Self-hosting

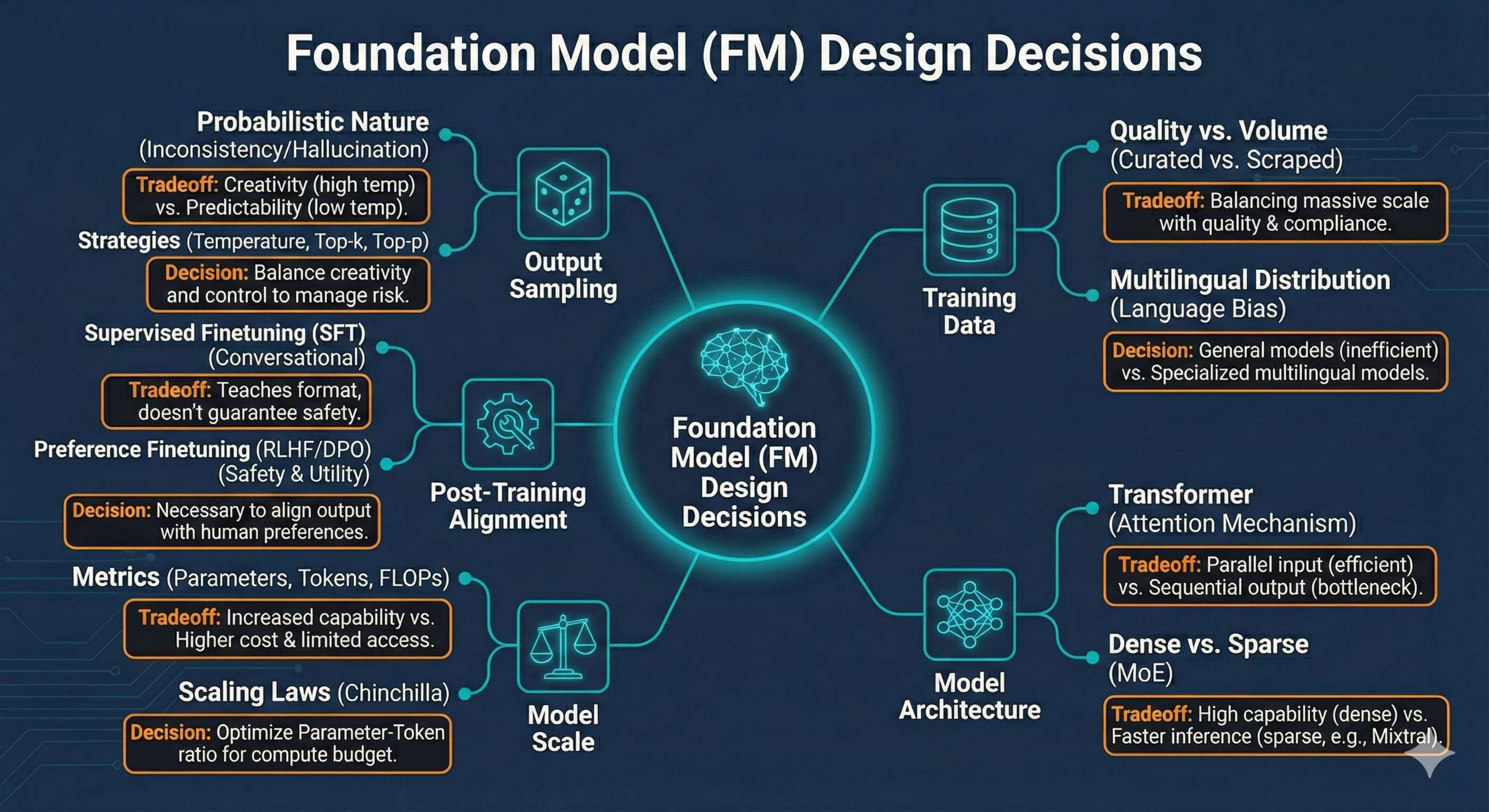
Key Enablers

Self-Supervision
(Infers labels from unlabeled data, e.g., Common Crawl)

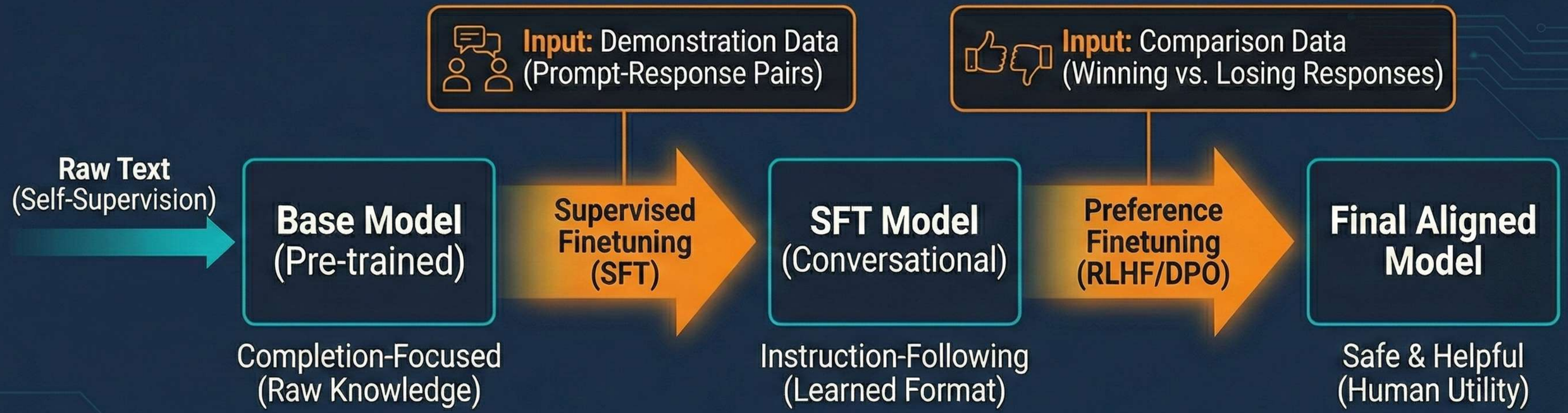
Tokens
(Efficient units balancing vocabulary size & semantics)

Shift to AI Engineering: Building applications atop readily available, powerful models

Foundation Model (FM) Design Decisions



The Post-Training Pipeline (From Raw to Aligned)



Analogy: The Finishing School

Pre-training = Reading every book (vast, raw knowledge).
Post-training = Finishing school (teaching manners, ethics, and appropriate application in conversation).

Evaluating Foundation Models: Complex, Open-Ended, and Critical

Challenges include: Sophistication, Open-Ended Outputs, Black Box models, Benchmark Saturation, and Lagging Investment.

Language Modeling Metrics (Predictive Accuracy)

- **Entropy (H):** Measures information content per token.
- **Cross Entropy (H(P,Q)):** Quantifies prediction difficulty against true data.
- **Perplexity (PPL):** Exponential of cross entropy; measures uncertainty (lower is better) and detects contamination.

Exact Evaluation (Objective & Functional)

- **Functional Correctness:** Does it perform the intended action? (e.g., pass@k for code).
- **Similarity Against Reference Data (Ground Truth):**
  - **Exact Match:** Only for definite, concise outputs.
  - **Lexical Similarity:** Measures text overlap (n-gram, BLEU, ROUGE).
  - **Semantic Similarity:** Measures closeness in meaning via embeddings (cosine similarity).

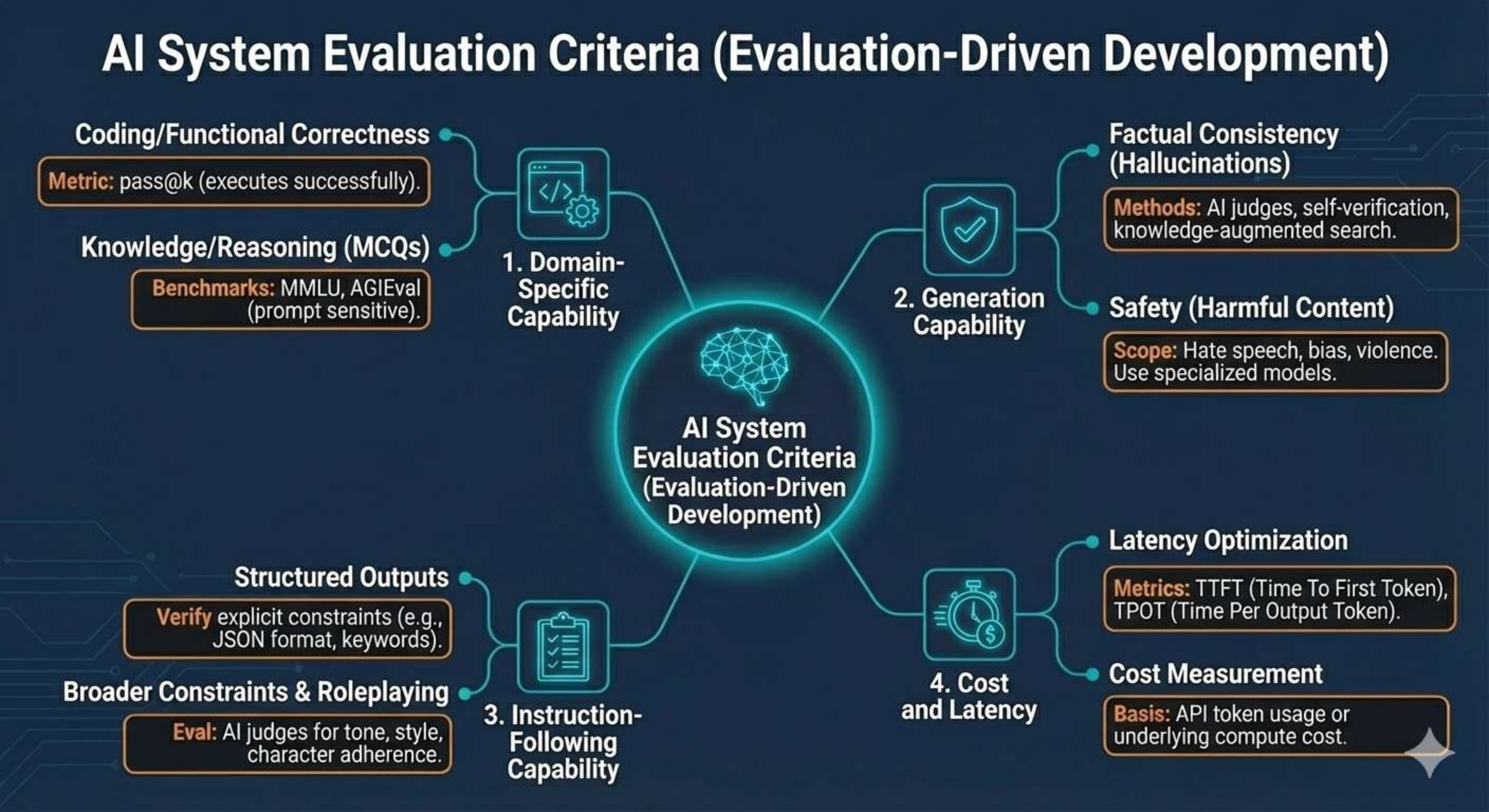
AI as a Judge (LLM-Based Automated Evaluation)

- **Rationale:** Fast, cheap, flexible for subjective quality without reference data.
- **Process:** Prompt judge with task, criteria, and scoring system.
- **Limitations:** Probabilistic inconsistency, criteria ambiguity, and biases (self-bias, verbosity bias).

Comparative Evaluation (Head-to-Head Ranking)

- **Mechanism:** Models compete in "matches"; evaluator chooses preferred output.
- **Ranking:** Results aggregated (e.g., Elo rating) for relative public ranking.
- **Advantage:** Easier than absolute scoring, avoids benchmark saturation, but shows only relative preference.

AI System Evaluation Criteria (Evaluation-Driven Development)



Model Selection Workflow: Balancing Quality, Cost, and Latency

Eliminate models based on: License, privacy policies, hardware compatibility.

Identify promising models. Wary of data contamination (benchmark data in training).

Rigorous tests with custom metrics & application-specific data.

Track performance, detect failures, gather feedback.

Model Decision: Buy (APIs) vs. Build (Self-Host)

Buy: Model APIs (e.g., OpenAI, Anthropic)

Pros: Ease of use, scaling, functionality (built-in guardrails).

Cons: Per-token costs, limited control, data privacy concerns.

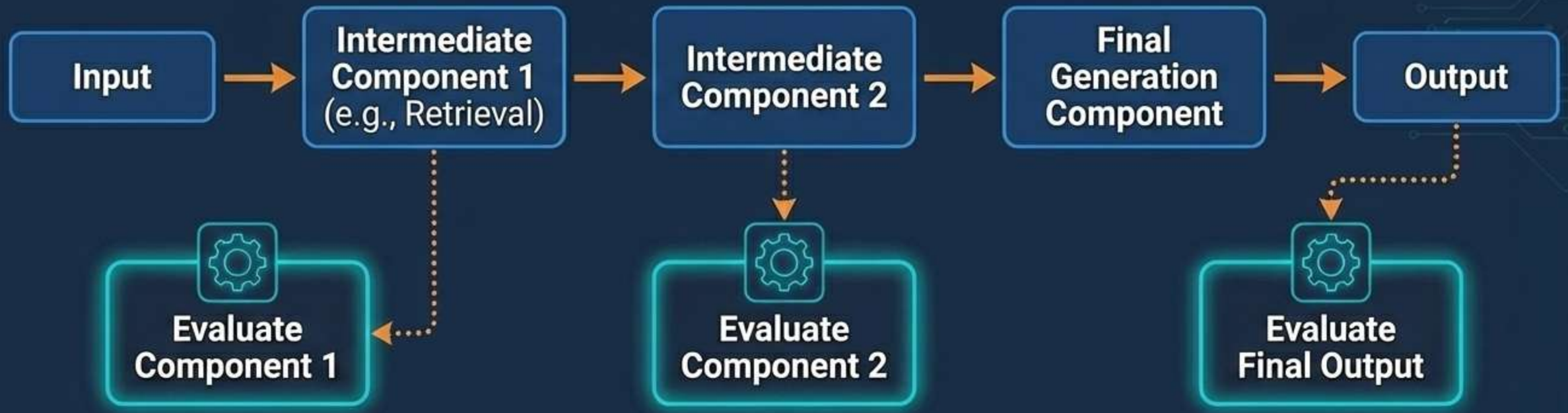
Build: Self-Host Open-Source (e.g., Llama, Mistral)

Pros: Full customization (finetuning), access to internals (logprobs), data privacy.

Cons: Significant engineering effort (deployment, optimization, maintenance).

shows only relative preference.

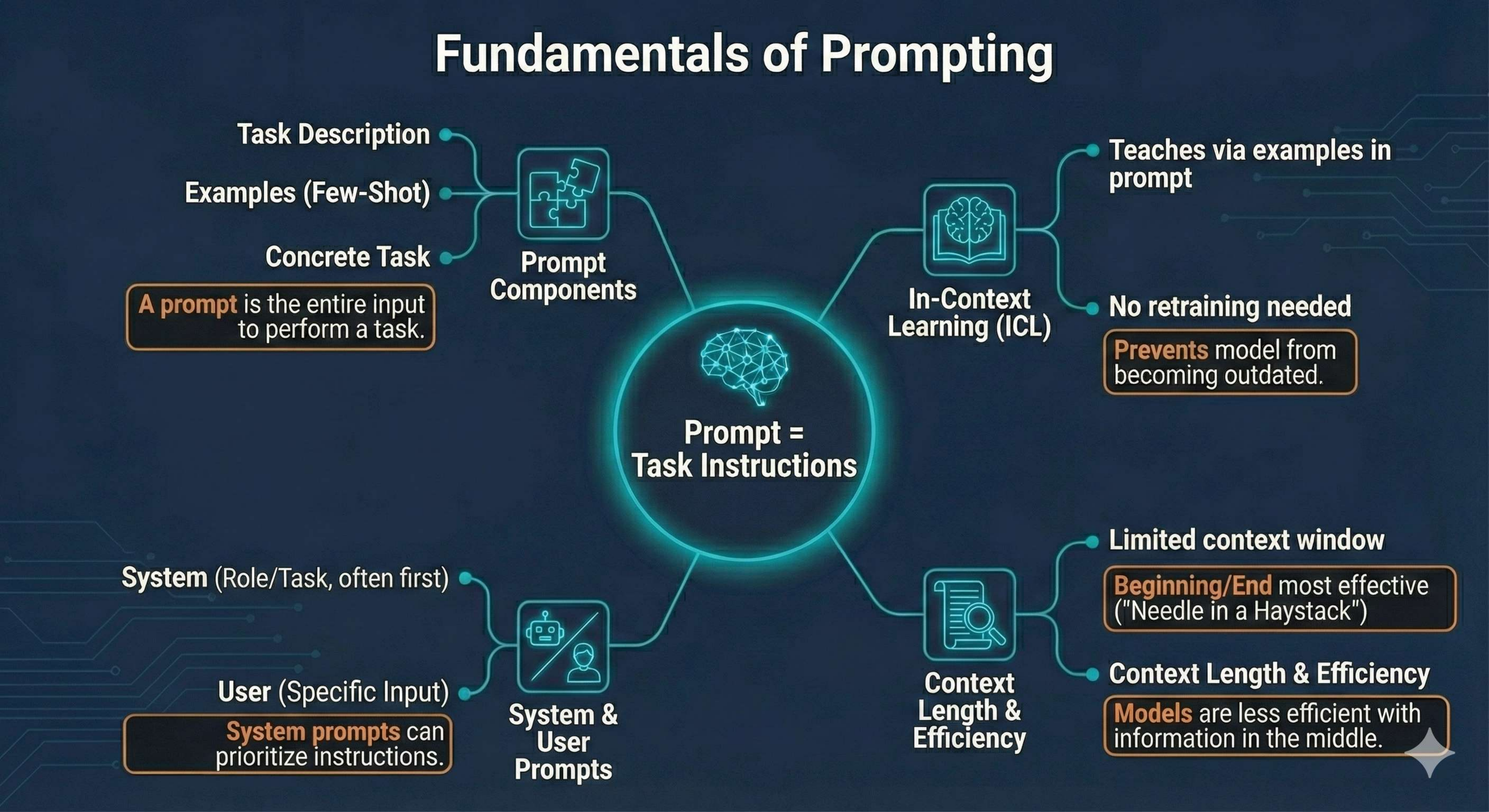
Comprehensive Evaluation Pipeline Design



Key Principles for Reliable Evaluation

- **1. Evaluate All Components:** Don't just test the final outcome; failures can be anywhere.
- **2. Create a Guideline:** Define criteria, scoring rubric, map performance to business metrics.
- **3. Define Methods & Data:** Mix classifiers & judges; explicitly slice data for fair assessment.

Fundamentals of Prompting



Prompt Engineering Best Practices

Clarity & Explicitness

- **Be unambiguous:** Specify exactly what to do, format, and scoring.
- **Adopt a persona:** Helps the model use the correct perspective.
- **Use examples:** Reduces ambiguity for the model.

Sufficient Context

- **Provide relevant data/docs:** Improves response quality.
- **Mitigate hallucinations:** Reduces reliance on potentially unreliable internal knowledge.

Decomposition & "Time to Think"

- **Break down complex tasks:** Simplifies subtasks, improves performance.
- **Chain-of-Thought (CoT):** Explicitly ask the model to “think step by step”.
- **Self-critique:** Prompt the model to check and revise its own output.

User Prompts

Iteration & Management

- **Continuous iteration:** Test to find optimal prompts.
- **Specialized tools:** Automate prompt creation, but verify performance and cost.
- **Version control:** Organize and version prompts separately from code.

Defensive Prompt Engineering: Risks & Defenses

Jailbreaking & Prompt Injection

- Bypass safety features (jailbreaking).
- Inject malicious instructions (direct or indirect via tools).

Reverse Engineering

- Deduce proprietary system prompts.
- Replicate or exploit the application.

Information Extraction

- Exploit model's memory.
- Extract sensitive or copyrighted information.

Model Vulnerability: Following Malicious Instructions

Models are trained to follow instructions, even malicious ones.

Model-Level Defense

- Train model to prioritize system instructions over conflicting user instructions.

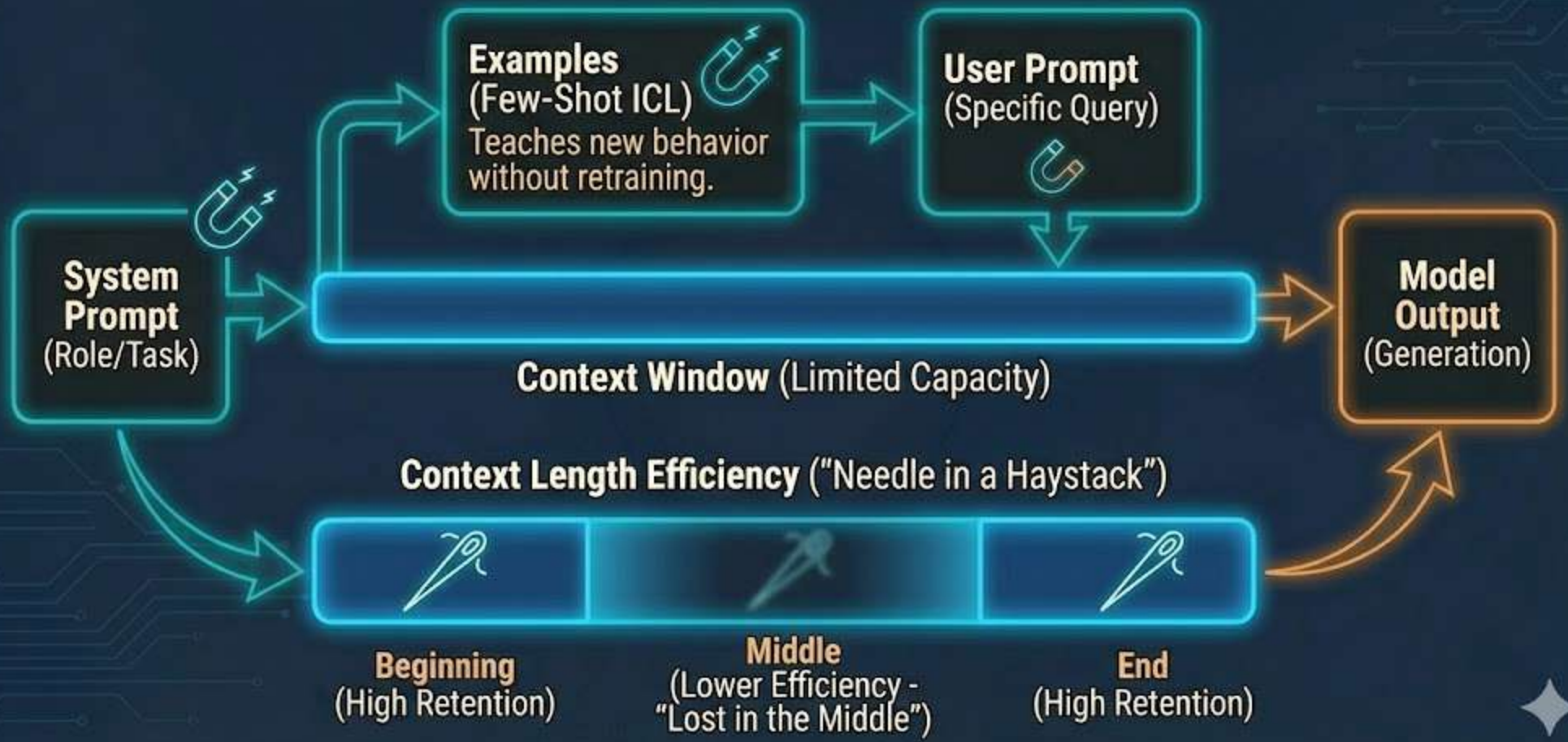
Prompt-Level Defense

- Repeat safety instructions.
- Make forbidden actions explicit.

System-Level Defense

- Isolate code execution (e.g., VMs).
- Add external guardrails and access controls.

In-Context Learning (ICL) & The Context Window



The RAG & Agents Landscape: Context & Action Patterns

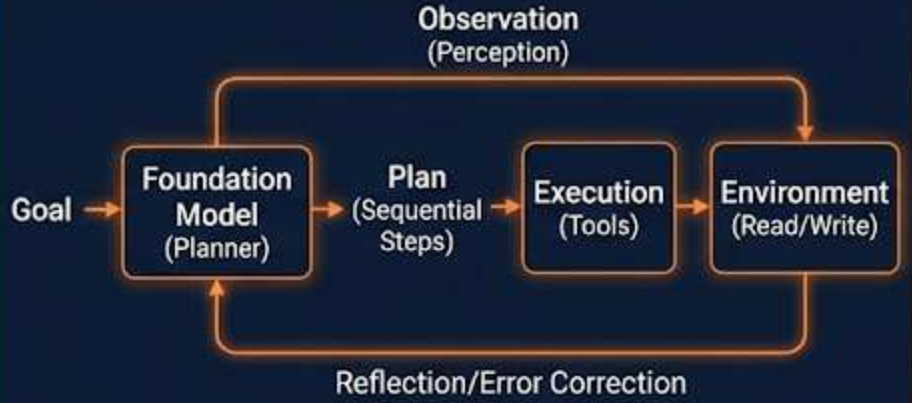
RAG (Retrieval-Augmented Generation)

Dynamic Context Construction
(Retrieve-then-Generate)

- **Enhances Quality** (External Memory)
- **Handles Extensive Knowledge**
- **Dominant Pattern**

Agents

Capabilities, Planning & Environment Interaction



- **Tool Inventory** (Knowledge, Capability, Write Actions)
- **Sophisticated Planning** (ReAct)
- **Autonomous & Workflow Automation**

RAG Architecture & Retrieval Algorithms

Basic RAG Architecture

Term-Based Retrieval (Sparse/Lexical)

Mechanism: Keyword overlap (lexical matching).

Pros: Faster, simpler, strong baseline.

Cons: Struggles with semantic ambiguity.

Algorithms: TF-IDF, BM25 (Inverted Index).

Hybrid Search & Reranking (Combinations)

Embedding-Based Retrieval (Dense/Semantic)

Mechanism: Numerical vectors (embeddings) & ANN search (e.g., HNSW, LSH).

Pros: Captures meaning/semantics, can outperform.

Cons: Higher computational overhead (generation & search).

Algorithms: CLIP, Annoy, Product Quantization.

RAG Architecture & Retrieval Algorithms

Basic RAG Architecture

Term-Based Retrieval (Sparse/Lexical)

Mechanism: Keyword overlap (lexical matching).

Pros: Faster, simpler, strong baseline.

Cons: Struggles with semantic ambiguity.

Algorithms: TF-IDF, BM25 (Inverted Index).

Hybrid Search & Reranking (Combinations)

Embedding-Based Retrieval (Dense/Semantic)

Mechanism: Numerical vectors (embeddings) & ANN search (e.g., HNSW, LSH).

Pros: Captures meaning/semantics, can outperform.

Cons: Higher computational overhead (generation & search).

Algorithms: CLIP, Annoy, Product Quantization.

Optimizing RAG & Memory Systems

Retrieval Optimization Tactics

Chunking Strategy

- Fixed size, recursive, overlap (prevents boundary loss).
- smaller chunks = diverse context.

Query Rewriting

- Reformulates ambiguous multi-turn queries for better accuracy.

Contextual Retrieval

- Augments chunks with metadata (tags, titles, summaries) for richer context.

Memory Systems (For RAG & Agents)

Short-Term Memory
(Context Window)
Fast access, limited capacity.
Stores immediate, session-specific info.

Long-Term Memory
(External Data / RAG Index)
Persistent, large-capacity.
Accessible via retrieval.

Internal Knowledge (Pre-training)
Inherent information learned during training.

Memory Management: Strategies (e.g., FIFO, summarization, entity tracking) to manage overflow.

Anatomy of an Agent: Components & Workflow

Foundation Model as Planner

- Generates plans, roadmaps of steps.
- Handles complex control flows (parallel, if/else, loops).

Planning & Reflection

- Decoupled from execution for validation.
- Uses patterns like ReAct for reasoning and acting.
- Evaluates outcomes.

Agent Tools (Inventory) - Augmenting Capabilities

Knowledge Augmentation (Read-only)

Examples: Text retrievers, SQL executors, web browsers.

Capability Extension

Examples: Calculators, code interpreters, text-to-image models.

Write Actions (Act upon environment)

Examples: Sending emails, placing orders, API calls (function calling).

The Concept of Finetuning: Adapting Foundation Models

Pre-trained Base Model
(General Knowledge)

Refines behavior, unlocks capabilities, improves sample efficiency.

Finetuning
(Transfer Learning)

Adjusts internal weights via further training.

Finetuned Model
(Task-Specific Capability)

1. Continued Pre-training
(Self-supervised)

Unlabeled, task-related data (e.g., legal documents).

2. Supervised Finetuning
(SFT)

High-quality annotated data pairs (instruction, response).

3. Preference Finetuning

Reinforcement learning with comparison data to maximize human preference.

When to Finetune vs. RAG: Behavior vs. Knowledge

Finetuning (Behavior-Based)

Primary Goal:
Form & Behaviour (Internal Adjustment)

Failure Mode Addressed:
Behaviour-based failures (e.g., not following format, irrelevant/unsafe responses)

Example:
Correcting a model to write compiling HTML or adhere to a specific SQL dialect.

RAG (Knowledge-Based)

Primary Goal:
Facts & Knowledge (External Supplement)

Failure Mode Addressed:
Information-based failures (e.g., lack of current/private knowledge, hallucination)

Example:
Supplementing a model to answer a specific product query with up-to-date specs.

Note: Techniques can be combined for maximum effect.

Finetuning Optimization: Overcoming the Memory Bottleneck

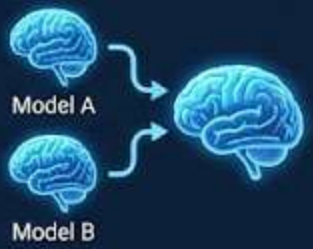
The Bottleneck:
Total Training Memory

1. Quantization (Compression)

Reduces precision (e.g., FP32 → FP16/INT8). Significantly lowers memory footprint.

2. PEFT (e.g., LoRA)

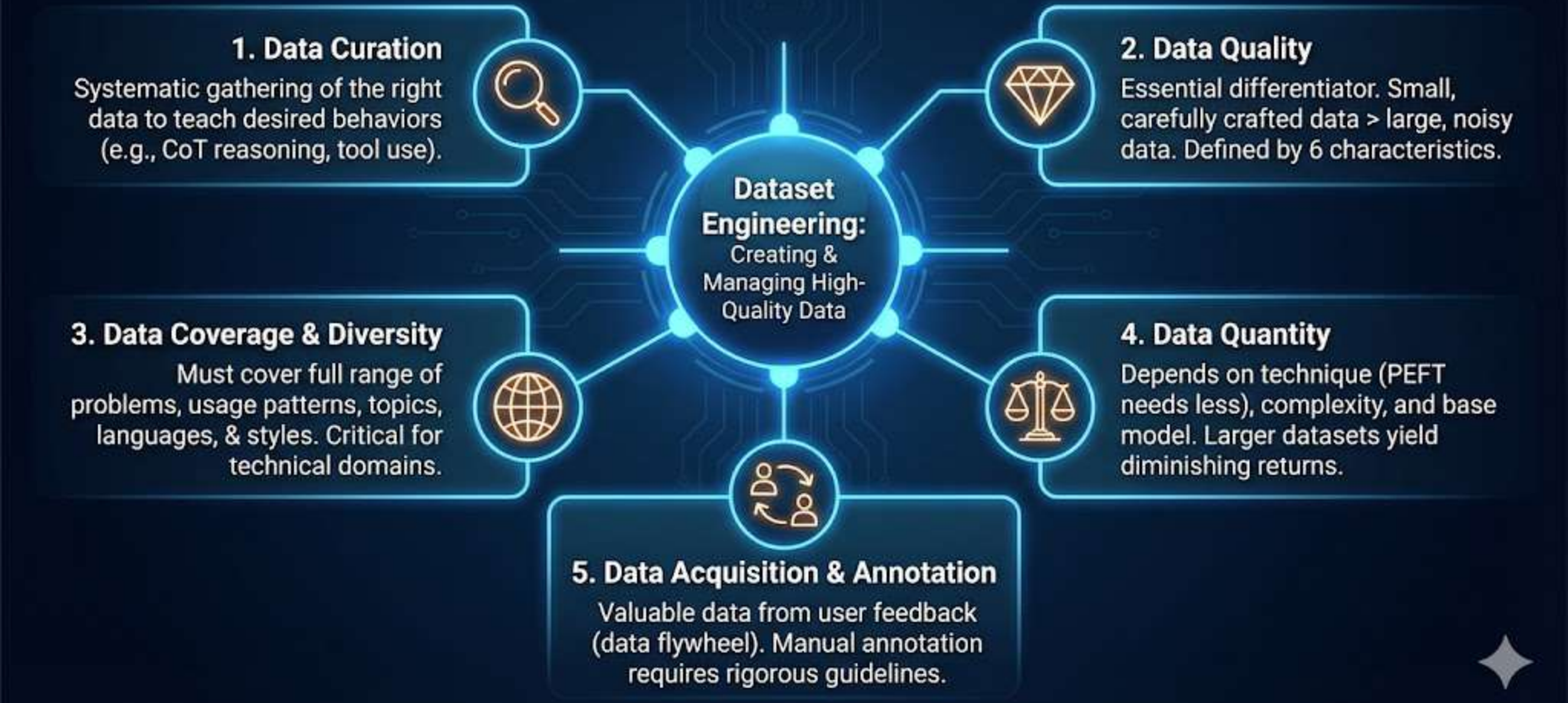
Updates only a small fraction of parameters (adapters). Keeps base model frozen.



3. Model Merging

Combines weights/adapters of multiple models into a single one. Enables multi-tasking.

Dataset Engineering Core Principles: The AI Differentiator



Finetuning Execution Workflow: A Step-by-Step Process

1. Base Model Selection

Start small, test with cheapest/fastest models.

Find optimal price/performance frontier before scaling.

2. Method Configuration (e.g., LoRA)

Select finetuning method (Full vs. PEFT).

PEFT (like LoRA) is often preferred for efficiency.

3. Hyperparameter Tuning

Optimize parameters.

Consider gradient accumulation for large batch sizes.

4. Data Requirement & Training

Prepare high-quality data.

Full: Millions.
PEFT: Hundreds/Thousands.

Beware of catastrophic forgetting.

The Six Pillars of Data Quality

1. Relevant

Examples must pertain to the specific task the model is being trained for.

2. Aligned with Task

Annotations should match task criteria (e.g., factual correctness, creativity).

3. Consistent

Uniform annotations across examples and annotators to avoid confusing the model.

4. Correctly Formatted

Follows expected format, free from extraneous tokens, errors, or inconsistencies.

5. Sufficiently Unique

Duplications managed to prevent skewing distribution or data contamination.

6. Compliant

Adheres to all privacy policies and regulations (e.g., no PII data).

Data Synthesis & Augmentation Techniques

Traditional (Rule-Based)

Templates & Transformations
Predefined templates, simple substitutions (synonyms, pronouns).

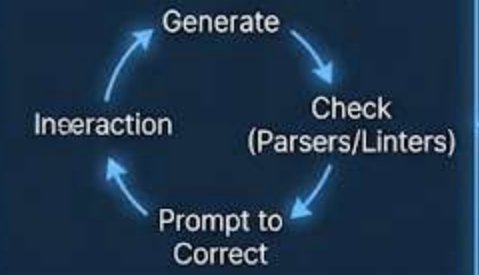
Use Case: Rare/dangerous scenarios (e.g., accidents).

AI-Powered Synthesis

Imitation & Paraphrasing
Translate to low-resource languages, paraphrase for diversity.

Instruction Synthesis
Generate new instructions/responses, reverse instruction approach.

Self-Correction Loop



Model Distillation
Train smaller student model on teacher-generated data for efficiency.

⚠ Limitations & Risks

- Superficial Imitation (Style over substance)
- Model Collapse (Irreversible defects from recursive training)
- Data Verification Challenges (Requires AI judges/rigorous tests)

The Data Processing Pipeline: From Raw to Ready

1. Inspection

Examine data via statistical analysis (e.g., token distribution plots) and manual review to identify patterns.

2. Deduplication

Eliminate redundant examples using similarity metrics (exact, fuzzy, semantic). Redundancy harms performance.

3. Cleaning & Filtering

Remove extraneous formatting (e.g., HTML tags). Filter out PII, sensitive, or toxic content for compliance.

4. Formatting

Adhere precisely to the required chat template and tokenization scheme. Incorrect formatting introduces errors.

Key Performance Metrics for Inference Services

Latency

- **TTFT (Time to First Token):** Prefill duration.
- **TPOT (Time per Output Token):** Generation speed.

Throughput

- Total tokens generated per second across all users.
- **Trade-off:** Batching increases throughput but can increase latency.

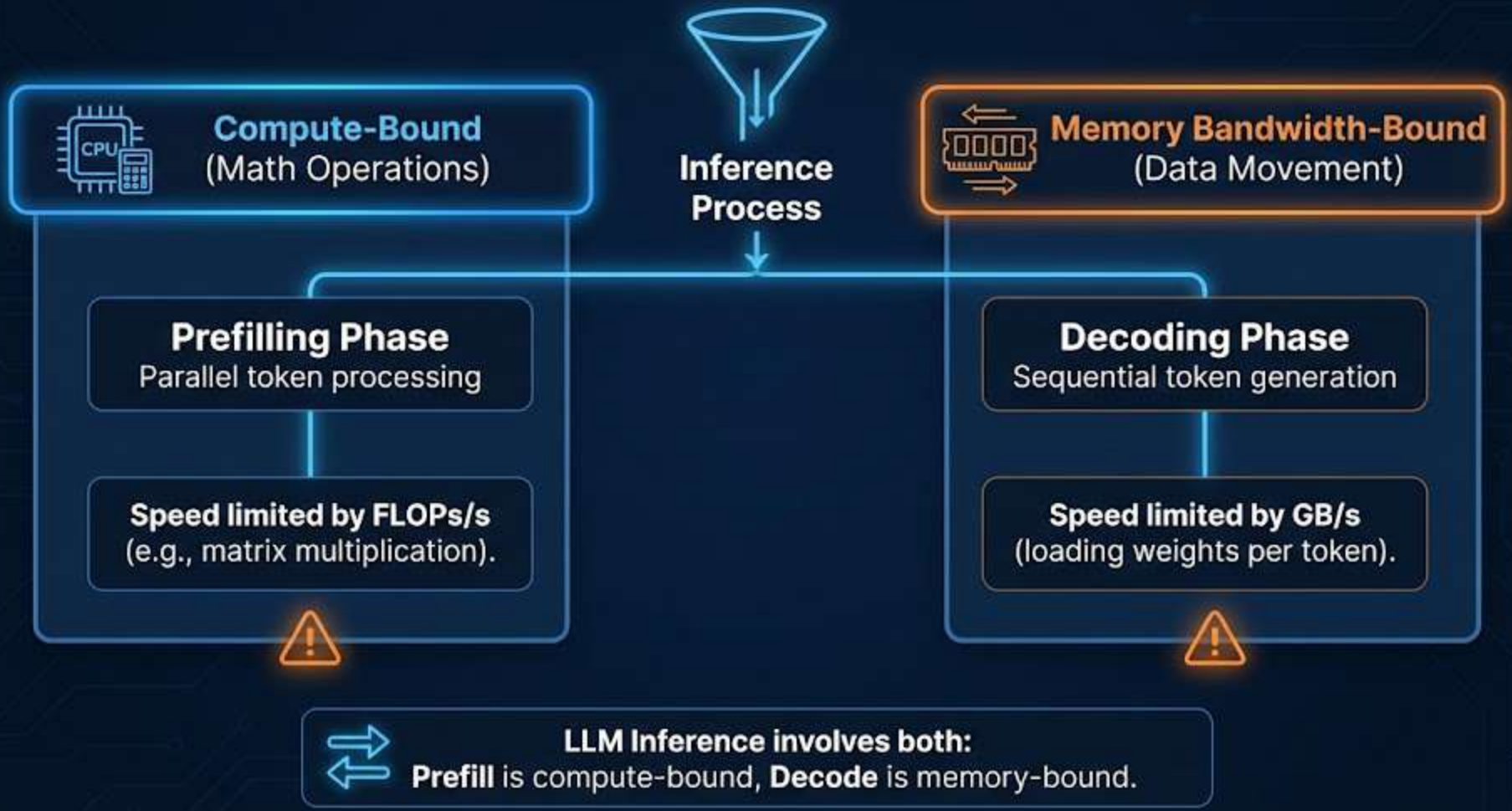
Goodput

- Requests per second meeting a defined SLO (Service Level Objective).

Utilization

- **MFU** (Model FLOP/s Utilization): Compute usage.
- **MBU** (Model Bandwidth Utilization): Memory bandwidth efficiency.

The Inference Bottleneck: Compute vs. Memory



Model-Level Optimization Techniques

1. Model Compression

Quantization (e.g., 16-bit → 4-bit), Distillation (train smaller student model).

2. Speculative Decoding

Draft model predicts tokens, target model verifies in parallel.

3. Parallel Decoding

Predict and verify multiple future tokens simultaneously (e.g., Medusa, Lookahead).

4. Attention Optimization

KV Cache redesigns (GQA, MQA) to reduce memory footprint.

Inference Service Optimization
(Resource Management)

Inference Service
(The Kitchen)

1. Batching (Continuous/In-flight)

Allows new requests to join a batch immediately, improving throughput (The faster "bus").

2. Decoupling Prefill & Decode

Disaggregates phases onto separate instances to prevent prefill from interrupting decoding.

3. Prompt Caching

Stores and reuses common prompt segments to reduce TTFT and cost (The "heated shelf").

4. Parallelism (Tensor & Replica)

Splits operators (Tensor) or creates model copies (Replica) to handle larger models/traffic.

Analogy: Improving Kitchen Management

Service-level optimization is like improving the kitchen's management—using faster buses, heated shelves, and specialized stations.