Getting your DeepSeek Models Production Ready

Abi Aryan

Abi Aryan
Founder, Abide AI
Book Author: LLMOps, GPU Engineering for AI Systems

About Me

Abi Aryan is the founder of Abide AI (www.abideai.com) and a machine learning research engineer with nearly a decade of experience building production-level ML systems.

A mathematician by training, she previously served as a visiting research scholar at the Cognitive Systems Lab at UCLA, under Dr. Judea Pearl, where she focused on developing intelligent agents.

Abi has authored research papers in AutoML, multi-agent systems, and large language models, and actively reviews for leading research conferences and workshops, including NeurIPS, TMLR (Transactions in Machine Learning Research), ACL (Association for Computational Linguistics), EMNLP (Empirical Methods in Natural Language Processing), and AABI (Advances in Approximate Bayesian Inference).

Author of two upcoming books: LLMOps *(Pre-orders available)*, GPU *Engineering for* AI *systems (First* 3 *chapters In Early Release soon)*

Currently advancing research in three key areas:

- Reflective intelligence for AI agents
- Distributed self-healing protocols for multi-agent systems
- GPU engineering for very large-scale AI systems

In today's talk, we will cover

01. Preparing the Model for Production

02. Deployment Environment Considerations

03. Testing, Validation, and Safe Production Rollout

01. Preparing the Model for Production

What does “Production Ready” Mean?

Is Robust

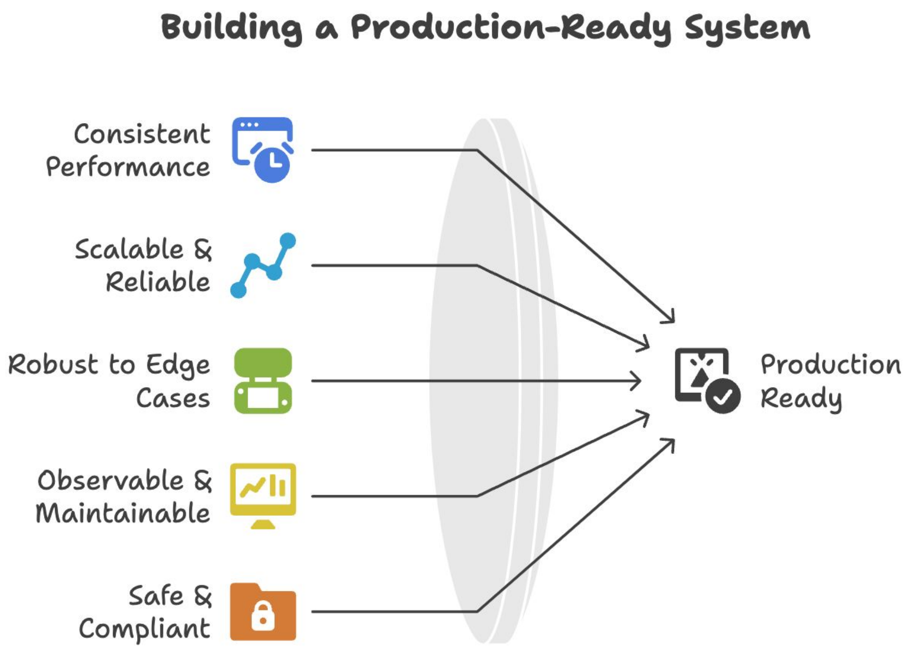
1. **Consistent performance** Meets latency, throughput, and accuracy targets under real-world load.
2. **Robust to edge cases** Handles unexpected inputs without crashing or degrading severely.

Is Reliable

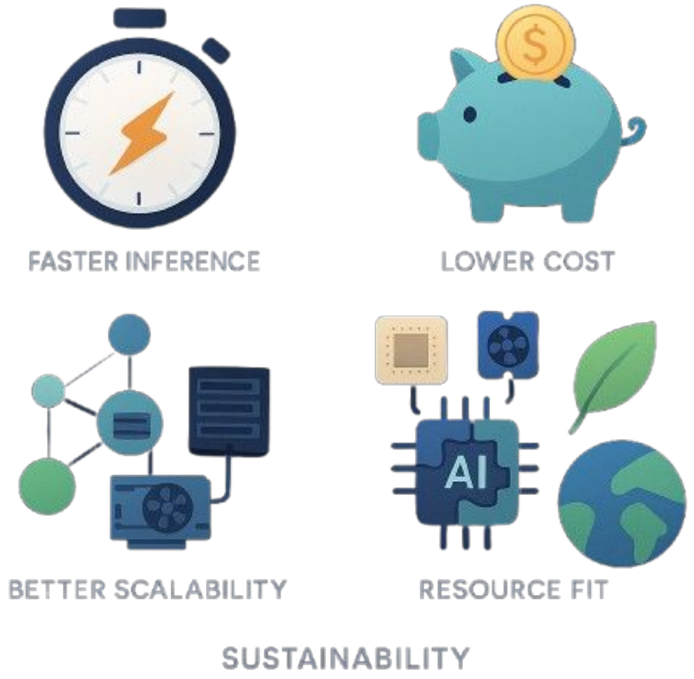
1. **Safe & compliant** Meets security, privacy, and ethical guidelines.

Is Scalable

1. **Scalable & reliable** Can handle traffic spikes, recover from failures, and run 24/7.
2. **Observable & maintainable** Has monitoring, logging, and metrics for quick troubleshooting.



So, we need to optimize it to make it "production ready"



What to optimize for?

Although the exact metrics would depend on the company and use-case, but generally speaking -

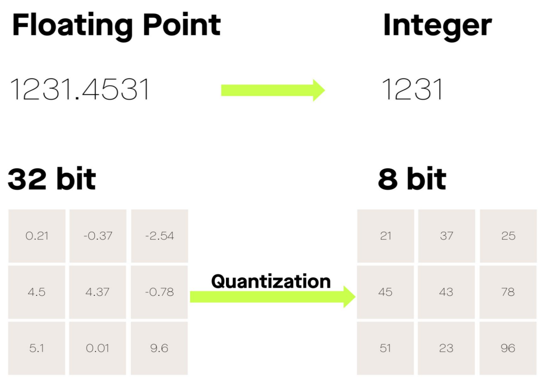
1. **Faster inference** → Lower latency improves user experience.
2. **Lower cost** → Reduced compute requirements cut infrastructure expenses.
3. **Better scalability** → Serve more requests with the same hardware.
4. **Resource fit** → Adapt the model to your available hardware (CPU, GPU, edge devices).
5. **Sustainability** → Less energy consumption for greener AI operations.

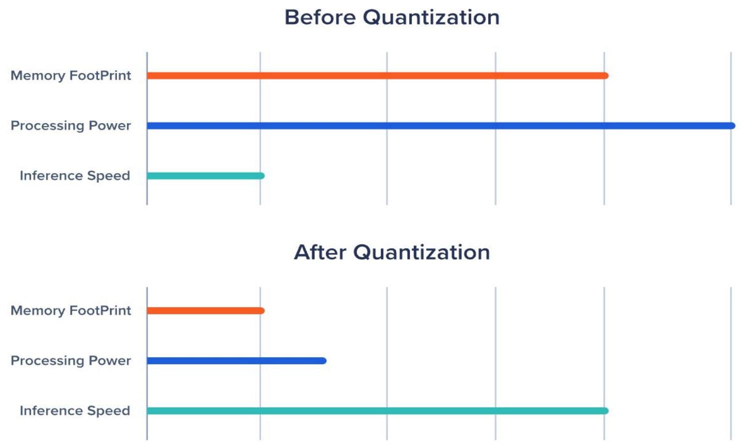
Model Optimization in Pre-Training: 3 approaches

1. Quantization

Reducing the precision of model weights (e.g., FP32 → INT8) without major accuracy loss.

Why? Smaller model size, faster inference, and lower memory usage.





Immediate Benefits:

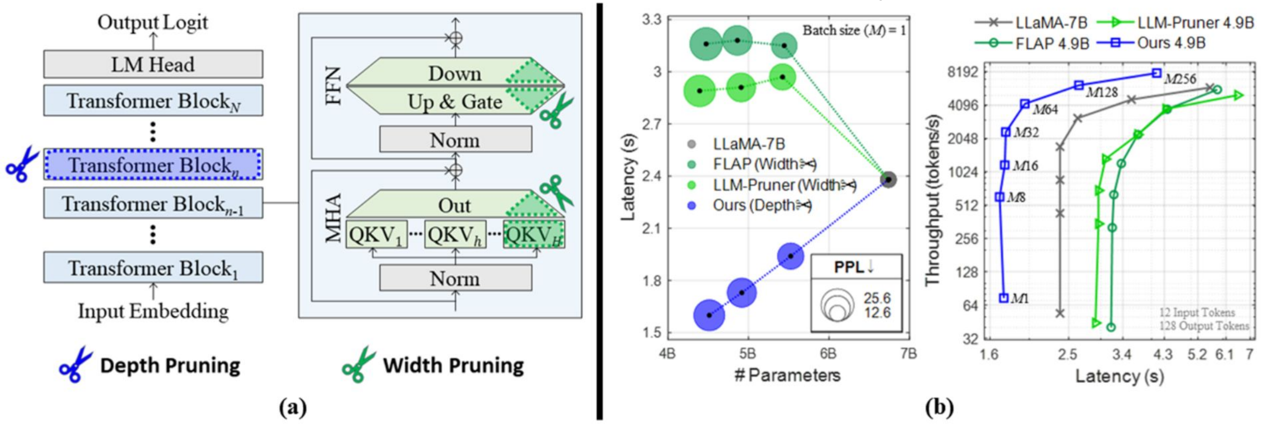
- *Runs on lower-end hardware.*
- *Reduced latency.*
- *Lower energy consumption.*

But tradeoff - slight drop in accuracy; may require calibration.

- Common formats: FP32, FP16, INT8, 4-bit quantization.

Model Optimization in Pre-Training: 3 approaches

By nota.ai team



2. Pruning

Removing redundant or less important weights/connections from a model.

- **Width Pruning**
- **Depth Pruning**

Why? Reduce size and speed up inference while keeping accuracy high.

Immediate Benefits:

- *Smaller memory footprint.*
- *Faster inference.*
- *Lower compute cost.*

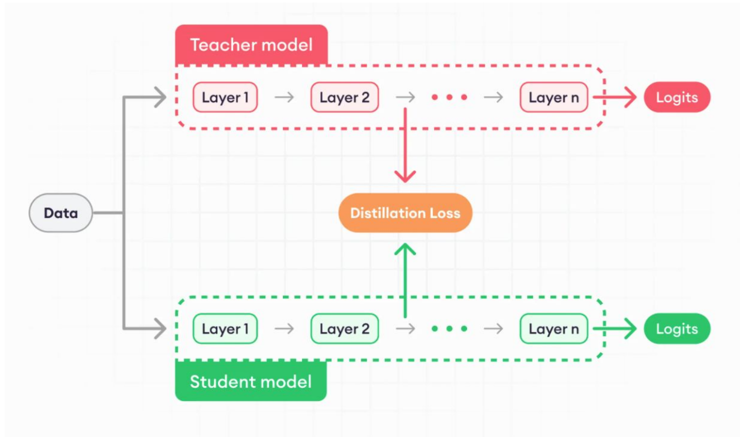
But obvious, trade-off: Risk of accuracy drop if over-pruned.

Model Optimization in Pre-Training: 3 approaches

3. Distillation

Training a smaller “student” model to mimic a larger, high-performing “teacher” model.

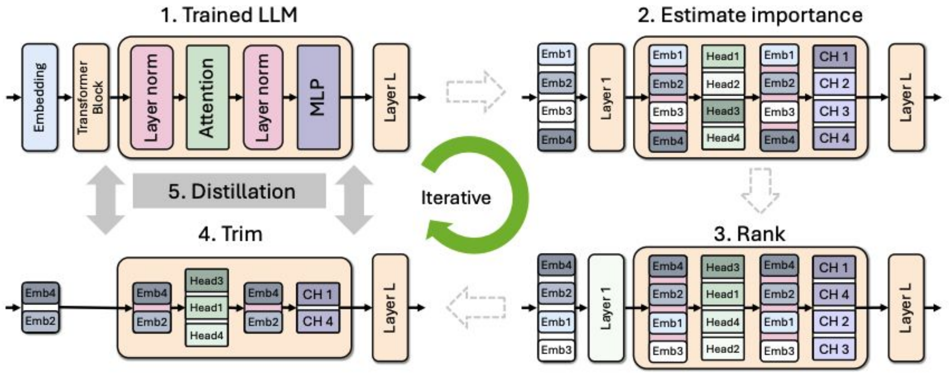
Why? Preserve accuracy while reducing size and inference cost.



Immediate Benefits:

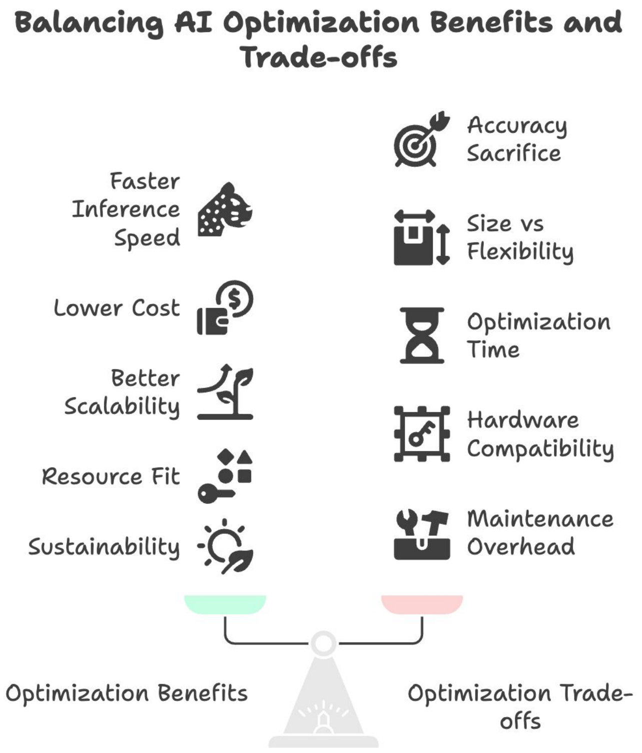
- *Smaller, faster, more deployable models.*
- *Retains much of the teacher’s accuracy.*

Use cases: Edge deployment, mobile apps, low-latency services.



arXiv: 2407.14679

But, every optimization comes with tradeoffs

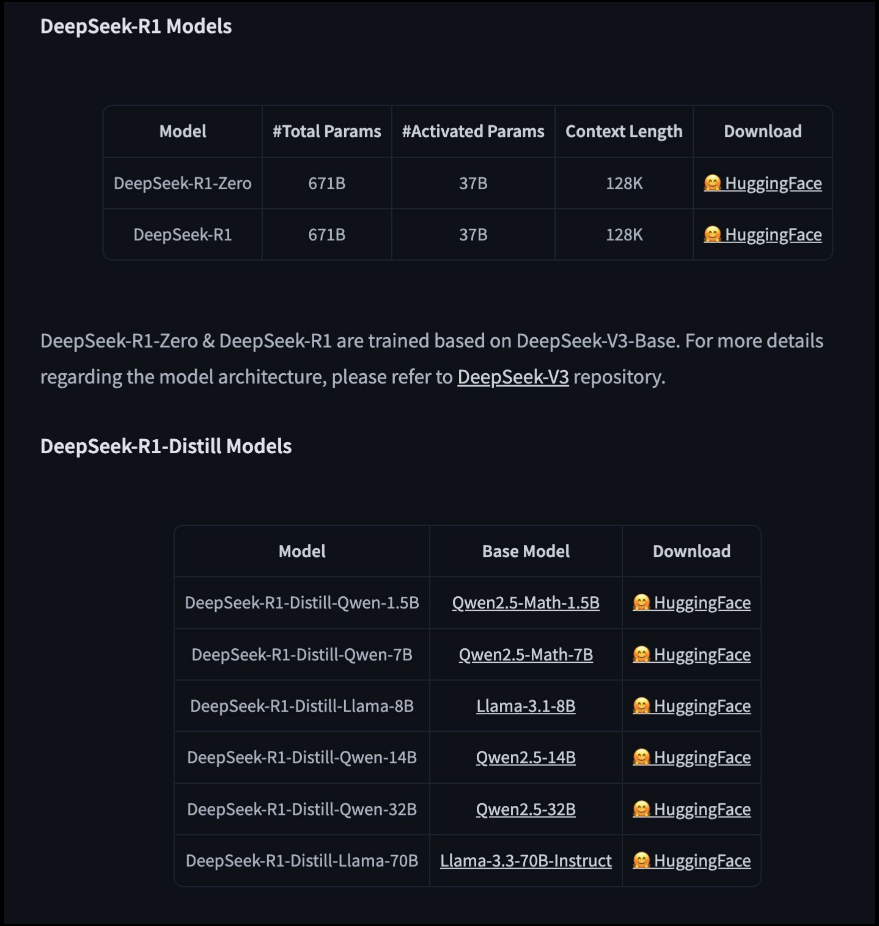


- ❖ Faster models may sacrifice some prediction quality.
- ❖ Smaller models may struggle with edge cases.
- ❖ Highly optimized models may be harder to retrain or update.
- ❖ Fine-tuning, quantization, or pruning takes engineering effort.
- ❖ Some optimizations work better on specific hardware (e.g., INT8 on GPUs with Tensor Cores).

The key is to measure, iterate, and align trade-offs with your deployment goals.

Use profiling, hardware-aware optimizations, and versioned experimentation to find the right balance.

Choose one? R1/V3/Distilled



DeepSeek-R1 Models

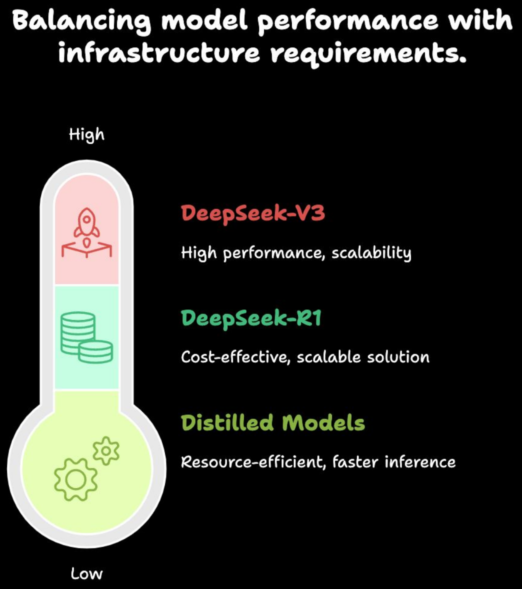
Model	#Total Params	#Activated Params	Context Length	Download
DeepSeek-R1-Zero	671B	37B	128K	🤗 HuggingFace
DeepSeek-R1	671B	37B	128K	🤗 HuggingFace

DeepSeek-R1-Zero & DeepSeek-R1 are trained based on DeepSeek-V3-Base. For more details regarding the model architecture, please refer to DeepSeek-V3 repository.

DeepSeek-R1-Distill Models

Model	Base Model	Download
DeepSeek-R1-Distill-Qwen-1.5B	Qwen2.5-Math-1.5B	🤗 HuggingFace
DeepSeek-R1-Distill-Qwen-7B	Qwen2.5-Math-7B	🤗 HuggingFace
DeepSeek-R1-Distill-Llama-8B	Llama-3.1-8B	🤗 HuggingFace
DeepSeek-R1-Distill-Qwen-14B	Qwen2.5-14B	🤗 HuggingFace
DeepSeek-R1-Distill-Qwen-32B	Qwen2.5-32B	🤗 HuggingFace
DeepSeek-R1-Distill-Llama-70B	Llama-3.3-70B-Instruct	🤗 HuggingFace

- DeepSeek-R1: balance between performance and cost, with 37 billion active parameters per request, making it suitable for large-scale deployments.
- DeepSeek-V3: Same parameter but utilizes a Mixture-of-Experts (MoE) architecture
- Distilled Models e.g., R1-Distill-Llama-70B


Balancing model performance with infrastructure requirements.
High
DeepSeek-V3
High performance, scalability
DeepSeek-R1
Cost-effective, scalable solution
Distilled Models
Resource-efficient, faster inference
Low

02. Deployment Environment Considerations

How to adapt the model for your environment?

01

Assess Hardware & Infrastructure

Evaluate CPU, GPU, TPU, memory, and storage constraints

02

Select Model Variant

Choose full, distilled, or MoE models based on targets

03

Configure Inference Settings

Optimize batch size, sequence length, and precision

04

Environment-Specific Optimizations

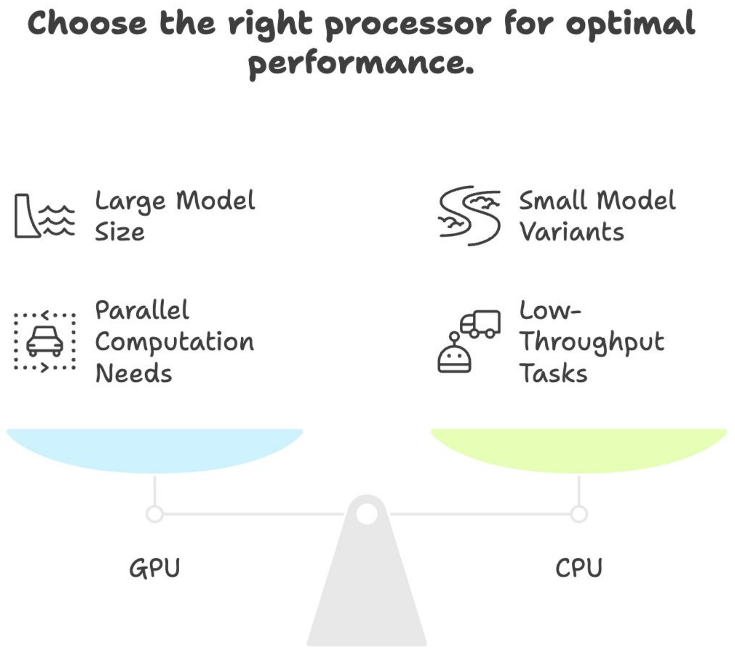
Tailor models for edge, cloud, or hybrid environments

05

Continuous Monitoring & Feedback

Track performance metrics and adjust settings

01/05. Hardware Selection


Choose the right processor for optimal performance.
Large Model Size
Small Model Variants
Parallel Computation Needs
Low-Throughput Tasks
GPU
CPU

GPU vs CPU:

R1: GPU recommended due to large model size and parallel computation needs.

V3: GPU preferred, but small V3 variants may run on CPU for low-throughput tasks.

Distilled R1-llama70b: Efficient enough for CPU in light workloads; GPU for higher throughput.

Edge vs Cloud:

Edge: Distilled models shine here; low latency, optimized resource usage.

Cloud: R1 or V3 can scale with distributed computation; better for high-capacity workloads.

02/05. Select Model Variant

R1 (Reasoning-first, large)

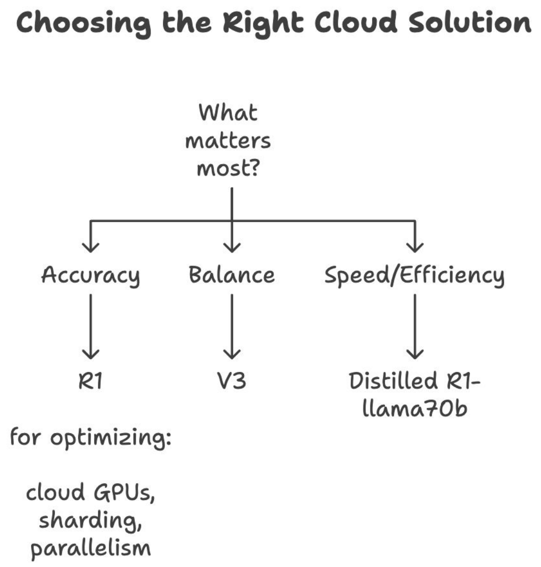
Best for: Complex multi-step reasoning, research-grade tasks, coding assistance, analytical workflows.

Environment Fit: Needs GPU clusters or cloud infra; benefits from model/pipeline parallelism.

V3 (Balanced generalist)

Best for: General-purpose chat, customer support, search, summarization, productivity apps.

Environment Fit: Runs well on GPUs, smaller variants even on high-end CPUs.



Distilled R1-llama70b (poor man's frnd)

Best for: Edge deployments, mobile/embedded apps, real-time inference (chat, autocomplete).

Environment Fit: Suitable for edge devices or smaller GPU/CPU setups.

03/05. Configure Inference Settings

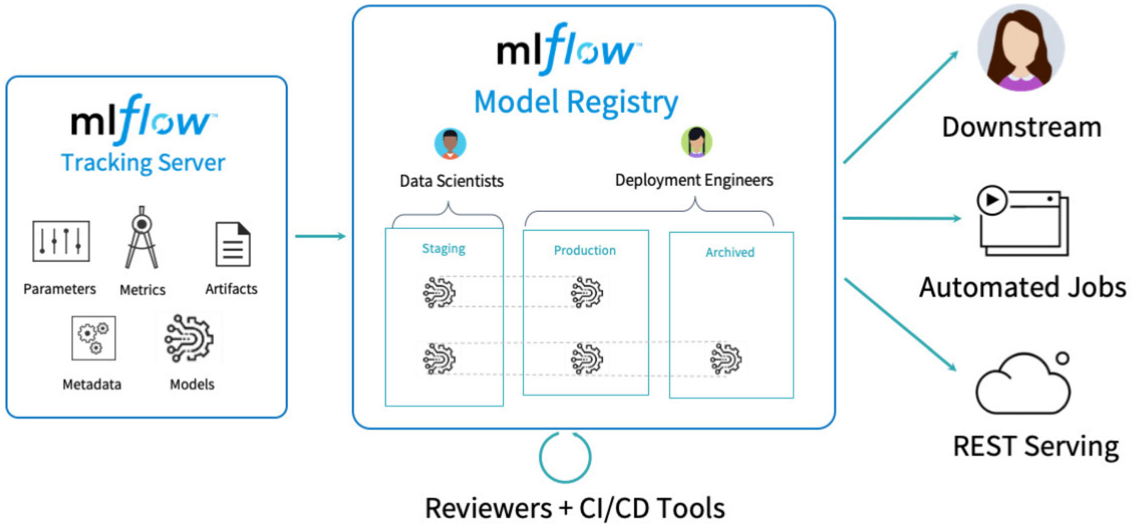
Setting	R1 (Large, Complex Reasoning)	V3 (Balanced Generalist)	Distilled R1-Llama70B
Scaling	Vertical scaling (multi-GPU, high RAM, A100/H100)	Horizontal scaling across mid-tier GPUs	Often runs on a single GPU or CPU
Batching	Larger batch sizes to keep GPUs busy	Dynamic batching for mixed workloads	Small batches, low-latency focus
Sharding	Needed (weights too large for single device)	**Sometimes needed for 70B+, else fine**	Not needed; fits smaller devices
Caching	Cache embeddings, frequent queries to save compute	Useful for repetitive workloads (search, Q&A)	Heavy caching less critical, model is fast
Elastic Scaling	**Cloud auto-scaling critical to manage costs**	Hybrid setups work well (edge + cloud)	Can run on edge; autoscaling optional
Latency vs Throughput	Prioritize throughput (batch-heavy workloads)	**Balanced tuning per application**	Prioritize latency (chatbots, realtime apps)

04/05. Environment Specific Optimizations

Dimension	**R1**	**V3**	**Distilled R1**
APIs & Interfaces	gRPC or streaming APIs to handle heavy payloads	REST/gRPC, flexible for mixed apps	REST APIs, lightweight SDKs for edge apps
Data Pipeline Compatibility	Requires strong preprocessing + normalization pipelines	Works with standard enterprise ETL pipelines	Minimal preprocessing; optimized for quick integration
Dependency Management	CUDA, NCCL, GPU-specific drivers; Docker mandatory	Standard ML frameworks (PyTorch, TensorFlow); Docker or Conda envs	Minimal deps; easy containerization; runs even on CPU Docker images
Security & Access Control	Strong auth + rate limiting (expensive inference calls)	Enterprise-grade auth; can integrate with IAM	Lightweight auth; edge cases may run locally with simple API keys
Monitoring Hooks	Needs detailed GPU/memory/latency metrics	Balanced observability (latency + quality metrics)	Lightweight monitoring; focus on availability rather than deep tracing
Fallback & Graceful Degradation	Route to smaller distilled model if overloaded	Switch to cached embeddings or fallback endpoint	Local fallback models or cached results (great for offline/edge)

05/05. Continuous Monitoring and Feedback

1. Track What You Deploy
2. Keep the Model Fresh
3. Update Safely
4. Watch Closely After Updates
5. Plan for the Full Lifecycle



03. Testing, Validation, and Safe Production Rollout

Testing it under real-world conditions: Stress Test!

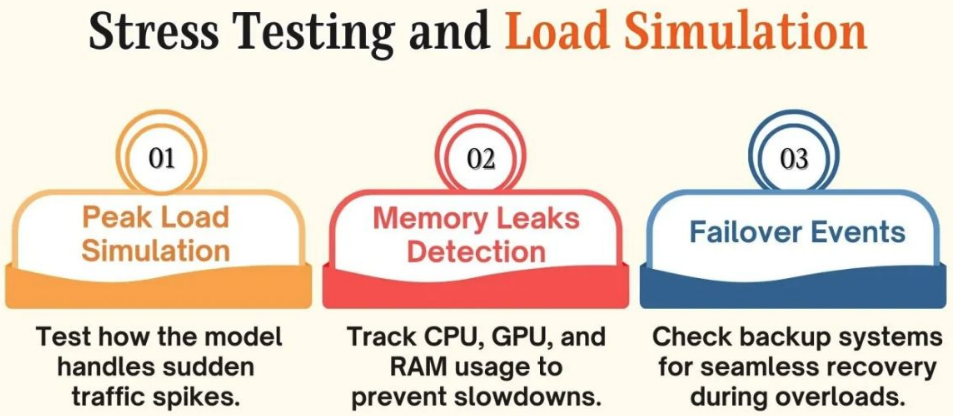
✅ **Synthetic vs Real-World Data**
Compare model performance on controlled datasets vs real input streams.
Identify edge cases that aren't captured in training data.

✅ **Latency & Throughput Stress Tests**
Test under peak load scenarios.
Measure system bottlenecks and resource limits.

✅ **Error Analysis & Robustness Checks**
Identify failure modes and mispredictions.
Track model degradation under unusual inputs or noise.

✅ **Scenario-Based Testing**
Simulate end-to-end workflows in your environment.
Include network fluctuations, API timeouts, and partial failures.

✅ **Automation & Repeatability**
Use testing frameworks or scripts for consistent benchmarking.
Enables comparison across model versions or deployment configurations.

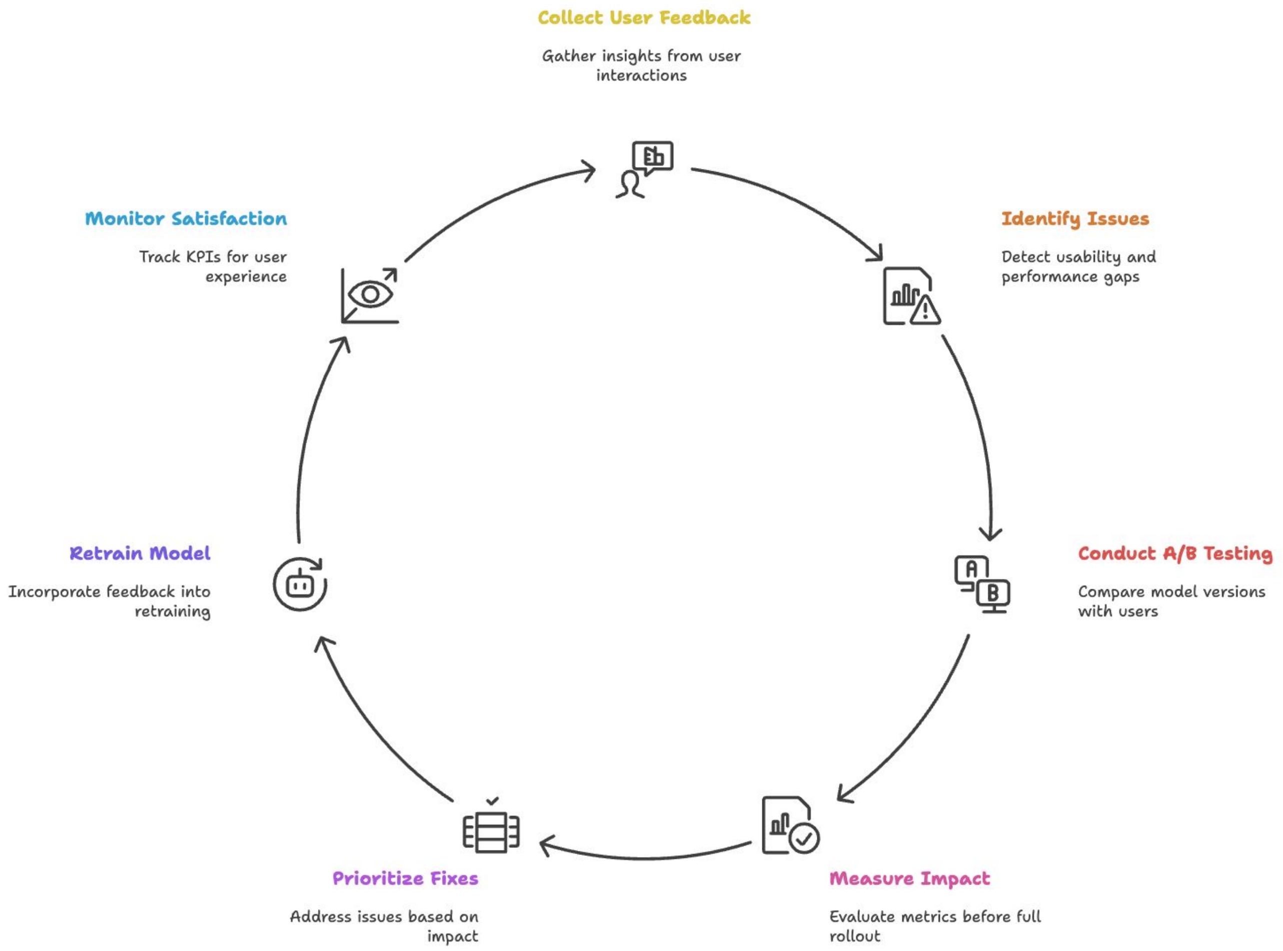

Stress Testing and Load Simulation

01 Peak Load Simulation
Test how the model handles sudden traffic spikes.

02 Memory Leaks Detection
Track CPU, GPU, and RAM usage to prevent slowdowns.

03 Failover Events
Check backup systems for seamless recovery during overloads.

Metrics & Evaluation


Collect User Feedback
Gather insights from user interactions
Identify Issues
Detect usability and performance gaps
Conduct A/B Testing
Compare model versions with users
Measure Impact
Evaluate metrics before full rollout
Prioritize Fixes
Address issues based on impact
Retrain Model
Incorporate feedback into retraining
Monitor Satisfaction
Track KPIs for user experience

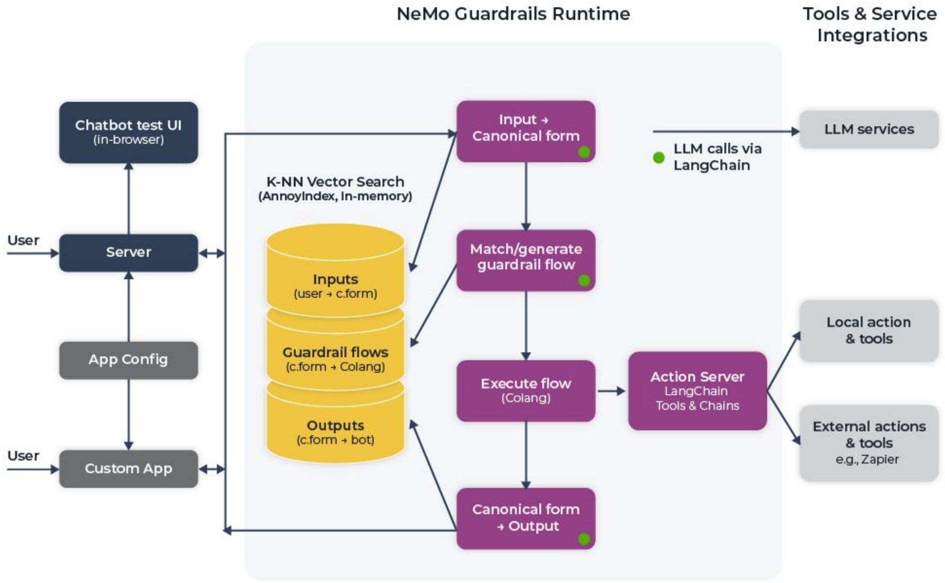
Guardrails and Safety

System Protection

- **Rate Limiting & Quotas** → Prevent overloads and abuse.
- **Input Validation** → Block malformed or malicious requests (e.g., prompt injection, code injection).
- **Fail-safes** → Gracefully degrade under high load (queue, shed, or fallback to smaller models).

Model Safety Controls

- **Output Filtering** → Block unsafe, biased, or policy-violating outputs.
- **Structured Guardrails** → Use frameworks (e.g., Guardrails.ai, NeMo Guardrails, RAIL specs) to enforce schema and policy.
- **Contextual Constraints** → Apply domain-specific safety layers (finance, healthcare, legal).


NeMo Guardrails Runtime
Tools & Service Integrations
Chatbot test UI (in-browser)
User
Server
App Config
User
Custom App
K-NN Vector Search (AnnoyIndex, In-memory)
Inputs (user → c.form)
Guardrail flows (c.form → Colang)
Outputs (c.form → bot)
Input → Canonical form
Match/generate guardrail flow
Execute flow (Colang)
Canonical form → Output
LLM calls via LangChain
LLM services
Action Server LangChain Tools & Chains
Local action & tools
External actions & tools e.g., Zapier

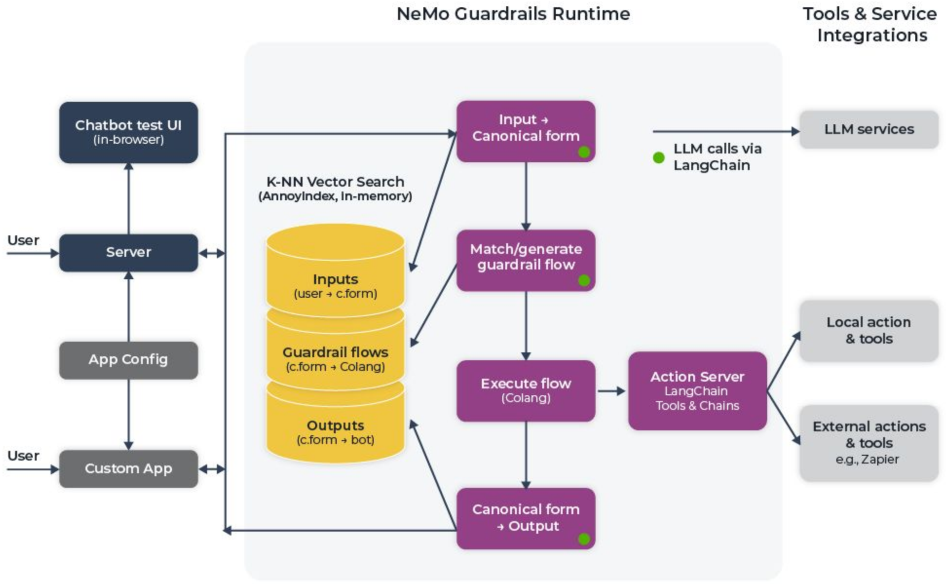
Guardrails and Safety

Real-Time Monitoring & Alerting

- **Anomaly Detection** → Flag drift, toxic responses, or unusual usage patterns.
- **Automated Alerts** → Notify teams on safety violations or latency spikes.
- **Closed-Loop Tracing** → Correlate request, model behavior, and downstream impact.

Resilience & Continuous Improvement

- **Fallback Mechanisms** → Default safe answers, cached responses, or human-in-the-loop escalation.
- **Incident Feedback Loops** → Feed safety incidents back into training or prompt refinement.
- **Iterative Hardening** → Continuously refine constraints, expand coverage, and test adversarial prompts.


NeMo Guardrails Runtime
Tools & Service Integrations
Chatbot test UI (in-browser)
User
Server
App Config
User
Custom App
K-NN Vector Search (AnnoyIndex, In-memory)
Inputs (user → c.form)
Guardrail flows (c.form → Colang)
Outputs (c.form → bot)
Input → Canonical form
Match/generate guardrail flow
Execute flow (Colang)
Canonical form → Output
LLM calls via LangChain
Action Server LangChain Tools & Chains
LLM services
Local action & tools
External actions & tools e.g., Zapier

Final Takeaways!

Optimization Matters - Use quantization, pruning, and distillation to balance speed, efficiency, and accuracy.

Adapt to Your Environment - Match hardware (GPU vs CPU), parallelization, and deployment patterns to your setup.

Scale & Integrate- Plan for batching, sharding, streaming inference, and smooth pipeline integration.

Test & Monitor in the Real World - Validate with real-world data, track metrics, and capture user feedback continuously.

Versioning & Safety First - Maintain version control, enable rollback, and apply guardrails for reliability.

Commit to Iterative Improvement - Refine models over time using feedback loops and operational insights.

Optimization Matters
Balance speed, efficiency, and accuracy using quantization, pruning, and distillation.
Match hardware, parallelization, and deployment patterns to your setup.
Adapt to Your Environment
Scale & Integrate
Plan for batching, sharding, streaming inference, and smooth pipeline integration.
Validate with real-world data, track metrics, and capture user feedback continuously.
Test & Monitor
Versioning & Safety
Maintain version control, enable rollback, and apply guardrails for reliability.
Refine models over time using feedback loops and operational insights.
Iterative Improvement

Q&A Time

@goabiaryan

Blogpost: ModelCraft

modelcraft.substack.com

Website: abiaryan.com