# Model Deployment Strategy Report

## Infrastructure Components

* Model Hosting: TorchServe / Triton / vLLM — Efficient model serving at scale.
* Web API Layer: FastAPI / Flask — Interface for HTTP-based model access.
* Load Balancer: Nginx — Distribute incoming traffic.
* Batching Gateway: vLLM — Dynamic batching for GPU optimization.
* Caching: Redis — Store frequently seen prompts/responses.
* Queue Management: Kafka — Handle overflow and queuing.
* Autoscaling: Kubernetes — Scale pods based on load.

## In-Flight Batching Strategy

* Use vLLM for token-level in-flight batching.
* Alternatively, create a custom batching queue that groups requests within a short window (e.g., 10-30ms).
  + Improves throughput.
  + Reduces GPU idle time.

## Caching Strategies

- Prompt-level Caching:

* Use Redis to cache repeated prompts and their outputs.
* Hash prompt text as cache key.

- Token-level Caching:

* Use key-value (KV) cache for transformer hidden states.
* Reduces computation for repeated tokens or long prompts.

## Model Optimization Techniques

ONNX - Optimized format for runtime inference

Quantization (INT8) - Reduce model size and memory bandwidth

FP16 / Mixed Precision - Speed up computation on supported GPUs

Model Sharding - Spread model across multiple GPUs

## Scaling Strategy

* Horizontal Scaling: Use Kubernetes to scale pods based on CPU/GPU usage.
* Vertical Optimization: Use high-performance GPUs (e.g., A100/H100).
* Request Routing: Sticky sessions or prompt segmentation for conversation consistency.

## Cloud Serverless Options

* Runpod
* Aws Lambda
* Google Vertex AI