Text-to-Video API – MVP & Production Design Document

Author: Keerthana Purushotham

Date: 2025-08-08

Purpose: This document outlines the design for a Kubernetes-deployed Text-to-Video API service using the Genmo Mochi-1 model to solve the problem of scalable, asynchronous, prompt-driven video generation.

1. Problem Statement (The Why)

Customers: Developers, researchers, and creative teams who need a scalable, programmatic text-to-video generation service.
Pain Points: Current video generation tools are often single-instance, blocking, and lack scalable API endpoints. Customers require asynchronous, concurrent, multi-GPU processing to handle high request volumes.
Urgency: Demand for generative AI video content is growing rapidly; this solution enables fast iteration and deployment.

2. Proposed Solution (The What)

Goal is to build an asynchronous text-to-video API using the Genmo Mochi-1 model hosted on an 8×H100 GPU Kubernetes worker node. The backend will handle job submission, tracking, and retrieval via JSON-based endpoints. A basic React-based frontend will allow prompt submission, status monitoring, and file downloads. The system will be deployed on Kubernetes (K8s) with GPU resource allocation, multi-replica redundancy, and horizontal scaling. Non-Goals: This MVP will not include advanced scheduling algorithms, RBAC, LLM-based load estimation, or zero-knowledge security layers - those are reserved for post-MVP.

3. Success Metrics (The How do we know it worked?)

1. MVP Success:
   - ≥95% job success rate.
   - P95 end-to-end latency ≤10 min.
   - Queue wait P95 ≤2 min.
   - Throughput ≥4 parallel jobs.
   - API availability ≥99% during demo.
   - 100% output artifact validity.
2. Production Success:
   - API availability ≥99.9%.
   - P95 latency ≤6 min, P99 ≤10 min.
   - Job retries <1%, DLQ <0.1%.
   - GPU utilization 70–90%.
   - Auth coverage 100%.
   - 0 critical CVEs in running images.

4. Open Questions & Assumptions

Considerations and Estimations:

1. Load visualization for video length vs prompt length:
    a. Img
2. Scale: Deployment patterns to prevent DoS by region, user-group etc with rollback, canary testing, retries, rate-limits etc.
3. Exceptions:
    a. Buggy prompt context from user – poor quality / lack of response
    b. Prompt work load exceeds resource allocation thresholds
    c. Infra security breaks -> retry / log relevant details
4. Are all tools compatible with potential upgrades and tool integrations without high refactoring costs? (ensure the OOPS aspects optimize computation without logical gaps or duplicate calculations
5. Concurrency handled by python orchestration over encapsulated, asynchronous Rust worker modules that run atomized request threads that close by virtue of Rust's memory/garbage management semantics that ensure that failed jobs do not break the validity of the session

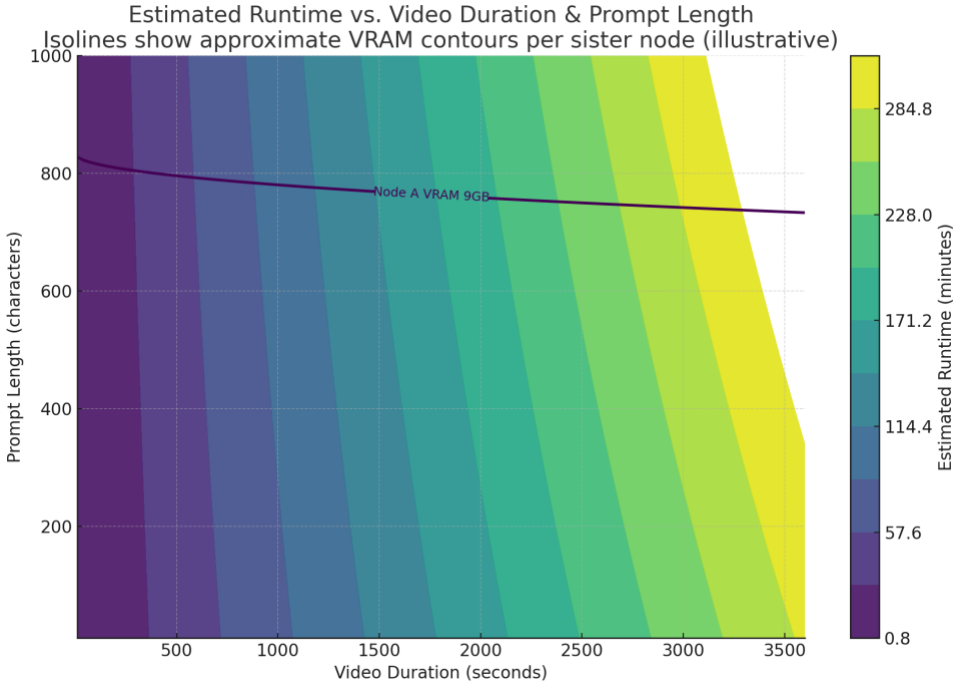
Assumptions:
- Video length ≤10s for MVP.
- Resolution ≤768p.
- API structure is REST over JSON.
- External object storage (S3/MinIO) is available.

Open Questions:
- Will the control plane ELB DNS be stable for external access? (known to cause costly DoS across regions resulting in downtime and loss)
- Expected concurrency limits at demo vs production scale?
- Any constraints on video length/quality &/or time limits from stakeholders?
- Complex multi-part prompts requiring state management, explicit network hardening (over sandboxing) plus encryption.

Other Corner Cases:

1. Pre-signed URL misuse / role changes mid-job / token skew.
    - recover any state from persistent storage and retry
2. Hot-keying in rate limiter; retry storms; DLQ loops.
    - (fastapi-limiter &/or redis). Link: stackoverflow & documentation for the caveat described below

- NOTE: FastAPI doesn't natively support this, but it's possible with a few libraries such the ones below, but will usually require some sort of database backing(redis, memcached, etc), although slowapi has a memory fallback in case of no database.
- https://pypi.org/project/fastapi-limiter/
- https://pypi.org/project/slowapi/
  ** In order to use fastapi-limiter, as seen in their documentation: You will need a running Redis for this to work.



3. Starvation of long jobs; convoy effects; head-of-line blocking.
   - dedicated long-task exception handler node with critical Alarm if task still fails;
   - some job length estimator module with dashboard tracking accuracy growth (for "is this potentially a long task based on context, linguistics, user/env metrics? Yes/No"
   - ….alarms at sev-2.5 if accuracy (true positives and negatives) consistently falls over time (false values are increasing. Check estimator logic), alarm at sev 2 if it falls immediately)
4. Log PII, high-cardinality labels; sampling hiding tail latency.
   - outliers and adversarial samples. Check if data corruption occurred via access/edit-logs, stack trace, etc to ensure no security exploitation broke the ML model.
5. Split-brain deploys across regions; partial rollbacks.
   - rollout
   - set
   - Scale
   - Autoscale
   - Auth
6. Model nondeterminism vs "golden" tests; flaky perf from noisy neighbors.

- Validation – check if things are right
- Sanity – ensure wrong things can't happen
- Unit – cover as many test cases, corner cases and outlier cases
- Integration – check if cross-tool features are ok
- Regression – ensure that new changes don't break existing functionality

5. Stakeholders & Next Steps

Key Stakeholders:
- Users: API consumers (developers, researchers)
- Tech Support: Handles incidents & outages
- Developers: Build & maintain backend/frontend
- Vendor Organization: Voltage Park infrastructure team
- Network Peers: Any API gateway/CDN providers
- Node Cluster: K8s worker node (8×H100)
- Control Plane: Managed by vendor, not directly accessible

Next Steps:
- Deploy initial API & worker pods on K8s.
- Implement asynchronous endpoints.
- **Finalize v1 prod features critical to release for enterprise scale.**
- Build basic React frontend.
- Integrate Prometheus/Grafana monitoring.
- Conduct load test for target throughput.
- Prepare for demo & stakeholder review.

Appendix: team input – Vote and choose v1 release features for prod.

Post-MVP Features – Prioritization Matrix

Purpose: Enable the team to quickly assess, vote, and sequence high-impact improvements after the MVP launch.
Voting Format: ✅ = must-have next, 🕒 = later, ❌ = not now.

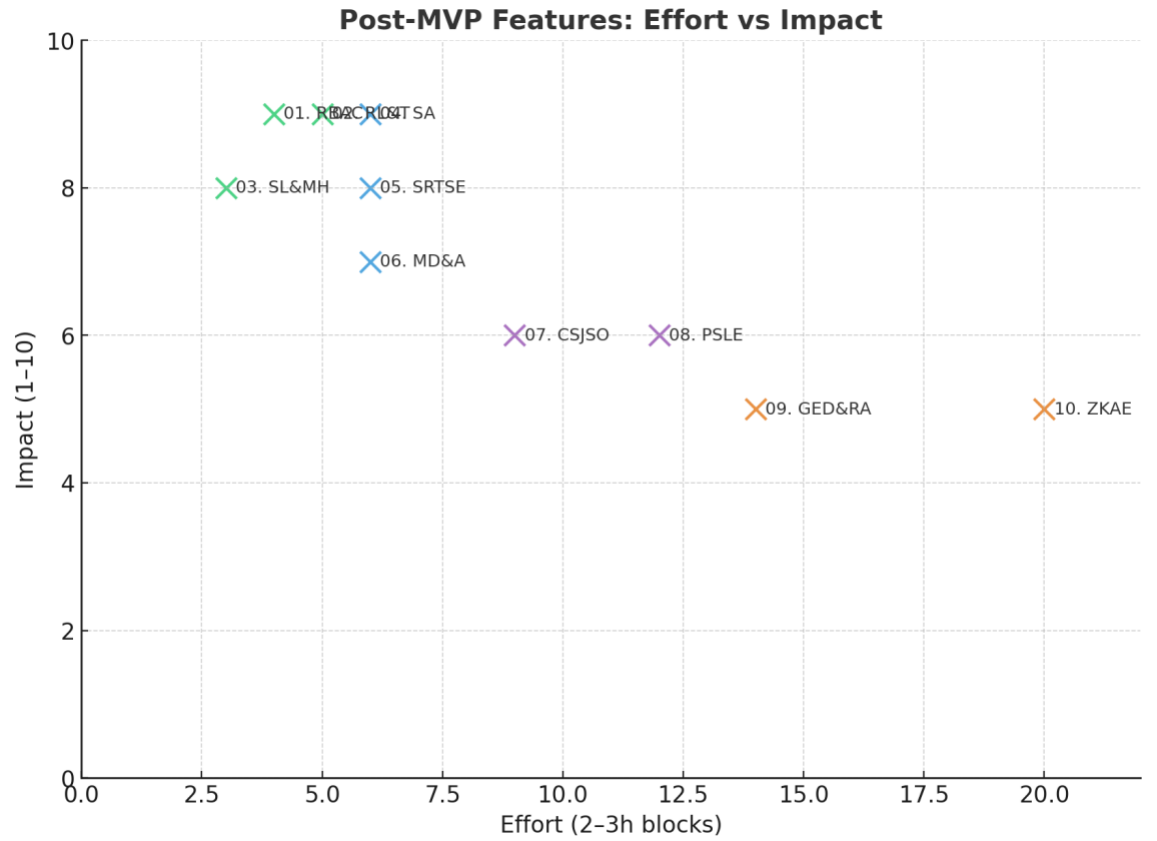
#	Acronym	Full Name	Stakeholders	Dependencies	Corner Cases	Effort (2–3h blocks)	Priority
☐01	RBAC	Role-Based API Access Control	user, dev, vendor org, control plane	Auth (04)	service-to-service calls, URL misuse	3–4	✅
☐02	RL&T	Rate Limiting & Throttling	user, tech-support, dev, vendor org, network peers	RBAC (01), Redis, Ingress	retry storms, hot keys	3–5	✅
☐03	SL&MH	Structured Logging & Monitoring Hooks	tech-support, dev, vendor org	none	PII leaks, log storms	2–4	✅
☐04	SA	Secure Authentication (JWT + OIDC)	user, dev, vendor org, control plane	IdP, RBAC	token expiry, revocation	4–6	✅
☐05	SRTSE	Stress & Regression Test Suite Expansion	dev, vendor org, node cluster	CI w/ GPU	flaky perf, model variance	4–6	🕒
☐06	MD&A	Minimal Dashboard & Alarms	user, tech-support, dev, vendor org	SL&MH (03)	alert fatigue, tenant leaks	4–6	🕒
☐07	CSJSO	Case-Specific Job Scheduling Optimizations	user, dev, vendor org, node cluster	job metadata	starvation, convoy	6–9	🕒
☐08	PSLE	Prompt-Specific Load Estimation	user, dev, vendor org	CSJSO (07)	bias, cold-start	8–12	❌
☐09	GED&RA	Global Edge Deployment & Rollback Automation	user, tech-support, dev, vendor org, control plane	MD&A (06), SL&MH (03)	split-brain, data residency	10–16	❌
☐10	ZKAE	Zero-Knowledge Architecture Expansion	user, vendor org, dev	KMS, TEE	key loss, debug blind	14–22	❌

Effort/Impact Visualization

> Documentation > Design.doc

u, 1 second ago | 1 author (You)

#	Acronym	Full Feature Name	Stakeholders	Dependencies
01	RBAC	Role-Based API Access Control	user, dev, vendor org, control plane	Auth (04)
02	RL\&T	Rate Limiting & Throttling	user, tech-support, dev, vendor, network peers	RBAC (01), Redis, Ingress
03	SL\&MH	Structured Logging & Monitoring Hooks	tech-support, dev, vendor org	none
04	SA	Secure Authentication (JWT + OIDC)	user, dev, vendor org, control plane	IdP, RBAC
05	SRTSE	Stress & Regression Test Suite Expansion	dev, vendor org, node cluster	CI w/ GPU
06	MD\&A	Minimal Dashboard & Alarms	user, tech-support, dev, vendor org	SL\&MH (03)
07	CSJSO	Case-Specific Job Scheduling Optimizations	user, dev, vendor org, node cluster	job metadata
08	PSLE	Prompt-Specific Load Estimation	user, dev, vendor org	CSJSO (07)
09	GED\&RA	Global Edge Deployment & Rollback Automation	user, tech-support, dev, vendor, control plane	MD\&A (06), SL\&MH (03)
10	ZKAE	Zero-Knowledge Architecture Expansion	user, vendor org, dev	KMS, TEE



I've mapped each feature into an **Effort vs Impact** matrix so it's easy to see trade-offs:

- **Green** = Immediate High-Impact / Low Effort (01, 02, 03)
- **Blue** = High-Impact / Medium Effort (04, 05, 06)
- **Purple** = Medium Impact / Higher Effort (07, 08)
- **Orange** = Niche Impact / High Effort (09, 10)