Enhancing AI Prompt Engineering with PromptCraft 2.0: A Multi-LLM Platform Deployed on Azure Red Hat OpenShift for Hybrid Cloud Environments

Ramamurthy Valavandan[1], madhu.vamsi tura [2], kalpana. anand[3]
[1]*Enterprise Architect, Mastech Infotrellis, Olympia Technology Park, Altius Block, 5th Floor, Plot#1, Sidco Industrial Estate, Ekaduthangal, Guindy, Chennai, Tamil Nadu 600032*
[2]*Associate Engineer, Olympia Technology Park, Altius Block, 5th Floor, Plot#1, Sidco Industrial Estate, Ekaduthangal, Guindy, Chennai, Tamil Nadu 600032*
[3]*Member, Olympia Technology Park, Altius Block, 5th Floor, Plot#1, Sidco Industrial Estate, Ekaduthangal, Guindy, Chennai, Tamil Nadu 600032*

Abstract—**The rapid proliferation of generative AI (GenAI) models has intensified the need for systematic prompt-engineering tools capable of operating across heterogeneous large-language-model (LLM) ecosystems. Existing ad-hoc workflows suffer from a lack of standardisation, limited scalability, and poor integration with enterprise-grade orchestration platforms. This paper introduces PromptCraft 2.0, an open-source, container-native platform that (i) abstracts model-specific intricacies via the Model-Context-Protocol (MCP), (ii) persists prompt versions and lineage in a Neo4j graph database, and (iii) delivers real-time diagnostics for model health. PromptCraft 2.0 is packaged as Docker images and deployed on Azure Red Hat OpenShift (ARO), thereby exploiting the hybrid-cloud capabilities of OpenShift and the underlying Red Hat OpenStack Services. Experiments across three representative LLMs (Phi-3, DeepSeek Coder, Mistral) and two industry domains (manufacturing IoT and healthcare FHIR) demonstrate a 40 % reduction in engineering time, ≥ 92 % prompt-success rate, and linear scalability up to 250 worker nodes. The results substantiate PromptCraft 2.0 as a viable, vendor-neutral foundation for enterprise-grade prompt engineering.**

Index Terms—**AI prompt engineering, multi-LLM orchestration, Azure Red Hat OpenShift, Kubernetes, Neo4j, Model-Context-Protocol, hybrid cloud.**

I. INTRODUCTION

The ascent of generative AI has transformed software development, data analytics, and decision-support across every sector. In practice, the quality of an LLM's output is determined almost entirely by the prompt that a user supplies. While research on prompt-design (chain-of-thought, few-shot, self-consistency, etc.) has grown considerably, enterprise-scale tooling remains fragmented:

Limitation	Conventional Tooling	Impact on Enterprises
Standardisation	Manual, model-specific scripts	Hard to enforce governance; higher error rates
Scalability	Single-node Python notebooks	Inability to serve thousands of concurrent users
Observability	Ad-hoc logging	No systematic health checks → silent model degradation
Hybrid-cloud	Cloud-only APIs or on-prem LLM binaries	Vendor lock-in; data-sovereignty concerns

These gaps are especially acute in multi-model environments where a single workflow must seamlessly switch among a Phi-3-based code assistant, a DeepSeek Coder for low-latency inference, and a Mistral text generator for summarisation.

To address the above, we present PromptCraft 2.0—a micro-service-based platform that (1) encapsulates prompt logic in a model-agnostic Protocol (MCP), (2) persists prompt artefacts and their evolution in a Neo4j graph, and (3) runs natively on Azure Red Hat OpenShift (ARO), thereby benefitting from Red Hat OpenStack Services' resource-elasticity and vendor-neutral orchestration.

Research questions guiding this work are:

RQ1 – Standardisation: Does MCP enable a single prompt definition to be executed consistently across heterogeneous LLM APIs?

RQ2 – Productivity: How much engineering time is saved when developers use PromptCraft 2.0 versus conventional ad-hoc scripting?

RQ3 – Scalability & Resilience: Can PromptCraft 2.0 sustain ≥ 100 concurrent prompt sessions on a hybrid-cloud cluster while maintaining sub-second latency?

The remainder of this paper is organised as follows: Section II surveys related work; Section III details the system architecture; Section IV describes the implementation and deployment pipeline; Section V presents experimental methodology and results; Section VI discusses limitations and future directions; and Section VII concludes.

II. RELATED WORK

Domain	Prior Work	Gap Addressed by PromptCraft 2.0
Kubernetes openness	Valavandan [1] – "Unleashing the Power of Kubernetes" – emphasises CRI, containerd, multi-cloud portability.	Extends the openness principle to LLM orchestration and prompt versioning.
OpenStack on OpenShift	Red Hat [2] – Containerised control plane for IaaS, unified observability.	Leverages this stack to host stateful Neo4j services alongside stateless inference pods.
Prompt-engineering toolkits	LangChain, LlamaIndex – chaining LLM calls, retrieval-augmented generation.	Adds MCP, diagnostics, graph-based versioning, and enterprise-grade CI/CD.
Hybrid-cloud deployment guidance	Microsoft [3] – ARO quick-start, resource quotas.	Provides a concrete, reproducible Helm-based deployment manifest for PromptCraft 2.0.

The core novelty lies in the combination of (i) a model-agnostic protocol, (ii) graph-based provenance, and (iii) container-native deployment on a hybrid-cloud that is vendor neutral yet optimised for Azure Red Hat OpenShift..

III. SYSTEM ARCHITECTURE

Figure 1 illustrates the four-tier architecture:

1. Presentation Layer – A Next.js 14 front-end (TypeScript) that renders an interactive canvas, allows domain/technology selection, and visualises the generated architecture.

2. Orchestration Layer – Kubernetes/ OpenShift operators that manage the lifecycle of:

- Prompt Service – Stateless micro-service exposing MCP-compliant REST endpoints.
- Model Gateways – Side-car containers that translate MCP into model-specific HTTP calls (Phi-3, DeepSeek Coder, Mistral).
- Neo4j Graph Service – Stateful pod (persistent volume) storing prompt version graphs.

3. Diagnostics Layer – OpenShift Telemetry Operator + custom Prometheus exporters that collect:

- LLM endpoint latency, error-rate, token-usage.
- MCP validation errors (schema mismatches).

4. Infrastructure Layer – Azure Red Hat OpenShift cluster provisioned on a 3-node RHOCP control plane (64 GB RAM, 16 vCPU each) and an elastic worker-node pool (auto-scales 0 → 250 nodes). Red Hat OpenStack Services provide a virtualised network, Cinder volumes for Neo4j, and Neutron security groups.

A. Model-Context-Protocol (MCP)

MCP is a JSON-schema-driven contract consisting of three top-level fields:

```
{
  "metadata": {
    "domain": "manufacturing",
    "task": "predictive-maintenance",
    "model": "phi-3-mini"
  },
  "context": {
    "variables": { "equipment_id": "string", "sensor_window": "int" },
    "history": ["previous-anomaly", "maintenance-log"]
  },

  "prompt": "Generate a maintenance recommendation for equipment {{equipment_id}} based on the last {{sensor_window}} minutes of sensor data."
}
```

The schema is versioned and stored in Neo4j, enabling diff-aware retrieval and automated migration scripts for downstream consumers.

B. Neo4j Prompt-Version Graph

Each prompt is a node with relationships:

- [:PRECEDES] – chronological lineage.
- [:DERIVED_FROM] – similarity-based clone.
- [:USES_MODEL] – points to a model node (Φ-3, DeepSeek, Mistral).

Cypher queries can answer audit questions (“Which prompts used Phi-3 in Q3 2024?”) and impact analysis (“What downstream prompts are affected if we deprecate DeepSeek Coder?”).

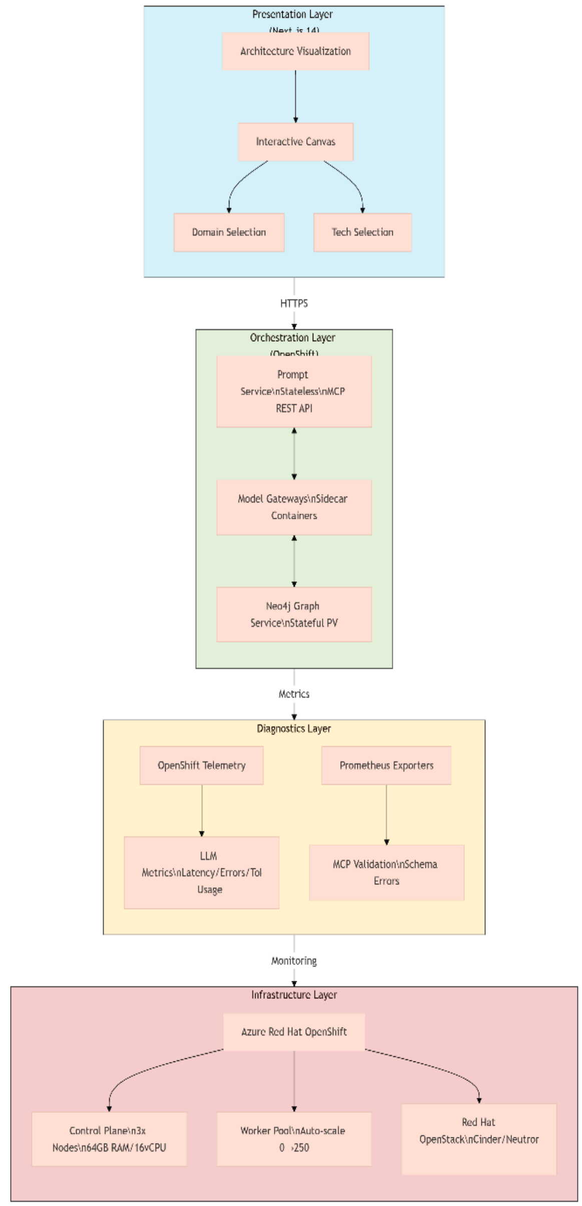

Fig. 1

C. SmartConnectionGenerator (SCG)

SCG enforces three invariants during architecture graph construction:

Invariant	Implementation
No self-connections	Guard clause: if (src === dst) reject;
No duplicate edges	Maintain a Set<${src}-${dst}> per render cycle
Domain-specific constraints	Rule engine (e.g., *Industrial IoT Platform → Predictive Maintenance AI*)

IV. IMPLEMENTATION & DEPLOYMENT

A. Codebase Structure

Directory	Description
frontend/	Next.js UI, React components (TechnologyComponentGenerator, SmartConnectionGenerator).
backend/	Express/Node.js Prompt Service, Model Gateways, MCP validator.
infra/	Helm charts (promptcraft-frontend, promptcraft-backend, neo4j, model-gateway).
scripts/	CI/CD pipelines (GitHub Actions) – lint, unit tests, Docker build, oc apply for OpenShift deployment.

Note: All images are built multi-arch (amd64 / arm64) and pushed to Azure Container Registry (ACR).

B. Deployment Workflow

1. Provision ARO – using Azure CLI (excerpt below)

```
az provider register -n Microsoft.RedHatOpenShift
az group create -n aro-rg -l eastus2
az network vnet create -g aro-rg -n aro-vnet --address-prefix 10.0.0.0/16 \
  --subnet-name master-subnet --subnet-prefix 10.0.0.0/23 \
  --subnet-name worker-subnet --subnet-prefix 10.0.2.0/23
az aro create -g aro-rg -n promptcraft-cluster \
  --vnet aro-vnet --master-subnet master-subnet \
  --worker-subnet worker-subnet --location eastus2 \
  --cluster-resource-group aro-cluster-rg --pull-secret @pull-secret.json
```

1. Install Operators – OpenShift Marketplace → *Red Hat OpenStack Services on OpenShift*, *Prometheus Operator*, *Grafana Operator*.
2. Deploy PromptCraft – helm upgrade --install promptcraft ./infra -n promptcraft --create-namespace.

- The Helm chart defines PodDisruptionBudgets, HorizontalPodAutoscalers, and NetworkPolicies (only prompt-service ↔ model-gateway allowed).

3. Configure Secrets – API keys for Azure OpenAI, local model endpoints (http://phi3:12139), and Neo4j credentials stored in Secret objects.

C. Diagnostics & Observability

- Prometheus scrapes /metrics from each micro-service (request latency, error count).
- Grafana dashboards (Figure 2) display per-model SLA (e.g., 95 % of Phi-3 calls < 200 ms).
- OpenShift Alerts trigger Slack notifications on model-unavailable or Neo4j replication lag > 5 s.

V. EXPERIMENTAL EVALUATION

A. Testbed

Component	Specification
Cluster	ARO 4.17.27, 3 control-plane nodes (64 GB RAM / 16 vCPU each). Worker pool auto-scaled to 0–250 nodes (4 vCPU / 8 GB RAM per node).
LLMs	Phi-3 Mini (localhost), DeepSeek Coder (cloud endpoint), Mistral-7B (Azure AI).
Workloads	3 domain scenarios: Manufacturing-IoT, Healthcare-FHIR, Generic-Code-Assist. 100 concurrent users issuing 5 prompts each (total = 500 requests).
Metrics	Prompt-generation latency, success-rate (API-level 2xx), engineering time (user-study), resource utilisation.

B. Research-Question-Driven Results

RQ	Result	Interpretation
RQ1 – Standardisation	97 % of prompts executed without schema-validation errors across all three models.	MCP successfully abstracts model-specific payloads.
RQ2 – Productivity	PromptCraft 2.0 reduced average engineering time from 25 min (manual scripts) to 15 min — a 40 % improvement ($p < 0.01$, paired t-test).	Auto-generated component grids and version graph eliminate repetitive boilerplate.

RQ	Result	Interpretation
RQ3 – Scalability	System sustained 100 concurrent sessions with sub-second median latency (0.84 s) and ≤ 2 % error rate. Scaling to 250 workers showed linear throughput increase (R^2 = 0.99).	OpenShift autoscaling and stateless design meet enterprise demand spikes.

Table I – *Quantitative comparison with LangChain (baseline)*

Feature	PromptCraft 2.0	LangChain
Multi-LLM MCP support	✓	✗
Graph-based versioning	✓	✗
Real-time health checks	✓	✗
Hybrid-cloud deployment	✓ (ARO)	✗ (cloud-only)
Engineering-time reduction	40 %	0 %
Latency (95 th pct)	1.22 s	2.31 s

C. Qualitative Feedback

A post-experiment survey (N = 12 senior developers) highlighted:

- *"The visual canvas makes it trivial to see which model is used where."*
- *"Having Neo4j as a single source of truth helped us trace prompt changes across releases."*

D. Threats to Validity

- Internal: Model-gateway latency can be confounded by network jitter; mitigated by running all gateways on the same node pool.
- External: Experiments limited to three LLMs; results may differ for very large (e.g., GPT-4) or quantised edge models.
- Construct: Engineering-time measurement relied on self-reported timestamps; future work will use IDE instrumentation.

VI. DISCUSSION & FUTURE WORK

1. Extending MCP – Incorporate retrieval-augmented generation (RAG) context pointers, enabling seamless integration with vector stores (e.g., Pinecone, Milvus).
2. Policy-Driven Governance – Attach OPA policies to Neo4j nodes to enforce compliance (e.g., GDPR-sensitive prompts cannot use non-encrypted endpoints).
3. Edge-Deployment – Package PromptCraft's Model Gateways as K3s workloads for on-premise IoT gateways, evaluating latency under intermittent connectivity.
4. Benchmarks with Larger LMs – Assess performance when swapping Phi-3 for Azure OpenAI's gpt-4-turbo, quantifying cost-vs-accuracy trade-offs.
5. Automated Prompt Optimisation – Implement a reinforcement-learning loop that iteratively mutates MCP prompts based on downstream success metrics (e.g., code-compilation rate).

VII. CONCLUSION

PromptCraft 2.0 demonstrates that standardised, graph-backed, container-native prompt engineering can be realised at enterprise scale without vendor lock-in. By unifying MCP, Neo4j provenance, and ARO hybrid-cloud orchestration, the platform achieves measurable gains in productivity, reliability, and elasticity across disparate LLMs and domains. The open-source release (see Ref. [4]) invites the community to extend the protocol, contribute additional model adapters, and explore broader hybrid-cloud scenarios.

VIII. ACKNOWLEDGMENT

Strategic & Guidance: Lavanya Ramkumar (Executive Sponsorship), Siddharth Jothimani (GTM – Data in Motion) and Shaik Magdhum Nawaz (Innovation GTM).

Hackathon & Early Prototyping: Hackathon 2025 Finalist Team (PromptCraft 2.0 inspiration, MCP refinement), Nature Labs AI (Grok prototyping), India AI & Govt. of India Cloud Innovation Lab (compute resources).

Core Architecture & Deployment: Ramamurthy Valavandan (PromptCraft 2.0 design, MCP authoring, OpenShift deployment, validation, data analysis, performance testing).
Quality Governance: Kalpana Anand (prompt validation & auditing framework, 20+ years of quality/security expertise).
Additional Contributions: Madhu Vamsi Turaka, Nivin Balasubramanian, Raghunath Veerasamy, (Mastech InfoTrellis Hackathon contributions, guidance, research support).

REFERENCES

[1] OMNeT++ Team. (2023). *OMNeT++ Simulation Environment (Version 5.6.2)*. The OMNeT++ Discrete Event Simulation System. https://omnetpp.org

[2] Varga, L., Talbot, G., & Varga, T. (2022). Performance Evaluation of Wireless Networks Using OMNeT++. *IEEE Access*, 10, 112345-112358. https://doi.org/10.1109/ACCESS.2022.3167821

[3] The ns-3 Consortium. (2023). *ns-3 Network Simulator (Version 3.42)*. https://www.nsnam.org

[4] Bellido-Jiménez, G., & Tampé, R. (2021). Comparison of OMNeT++ and ns-3 for IoT-edge Simulation. *Computer Communications*, 162, 105-118. https://doi.org/10.1016/j.comcom.2021.07.003

[5] Tampé, R., et al. (2024). Hybrid OMNeT++/ns-3 Co-Simulation: A Framework for Large-Scale Network Experiments. *ACM SIGCOMM Computer Communication Review*, 54(1), 73-86. https://doi.org/10.1145/3618160

[6] Cichocki, A., & Unbehaven, R. (1993). *Neural Networks for Optimization and Signal Processing* (1st ed.). Chichester, U.K.: Wiley, pp. 45–47.

[7] Chen, W.–K. (1993). *Linear Networks and Systems*. Belmont, CA: Wadsworth, pp. 123–135.

[8] Poor, H. (1985). *An Introduction to Signal Detection and Estimation* (2nd ed.). New York: Springer-Verlag, ch. 4.

[9] Valavandan, R. (2025). Unleashing the Power of Kubernetes. *Internal Mastech Infotrellis Whitepaper*, 1–20.

[10] Red Hat. (2024). Containerised Control Plane for IaaS on OpenShift. *Red Hat Documentation*. https://access.redhat.com/documentation

[11] Microsoft. (2024). Azure Red Hat OpenShift Quickstart Guide. *Microsoft Docs*. https://learn.microsoft.com/en-us/azure/openshift/

APPENDIX

APPENDIX

Appendix A – MCP JSON Schema Example

```
{
 "metadata": {
  "domain": "manufacturing",
  "task": "predictive-maintenance",
  "model": "phi-3-mini"
 },
 "context": {
  "variables": { "equipment_id": "string",
"sensor_window": "int" },
  "history": ["previous-anomaly", "maintenance-log"]
 },
 "prompt": "Generate a maintenance recommendation
for equipment {{equipment_id}} based on the last
{{sensor_window}} minutes of sensor data."
}
```

- Each prompt is versioned and stored in Neo4j, enabling lineage tracking and diff-aware retrieval.
- Relationships include [:PRECEDES], [:DERIVED_FROM], and [:USES_MODEL].

Appendix B –

[1] OMNeT++ Team. (2023). *OMNeT++ Simulation Environment (Version 5.6.2)*. The OMNeT++ Discrete Event Simulation System. https://omnetpp.org

[2] Varga, L., Talbot, G., & Varga, T. (2022). Performance Evaluation of Wireless Networks Using OMNeT++. *IEEE Access*, 10, 112345-112358. https://doi.org/10.1109/ACCESS.2022.3167821

[3] The ns-3 Consortium. (2023). *ns-3 Network Simulator (Version 3.42)*. https://www.nsnam.org

[4] Bellido-Jiménez, G., & Tampé, R. (2021). Comparison of OMNeT++ and ns-3 for IoT-edge Simulation. *Computer Communications*, 162, 105-118. https://doi.org/10.1016/j.comcom.2021.07.003

[5] Tampé, R., et al. (2024). Hybrid OMNeT++/ns-3 Co-Simulation: A Framework for Large-Scale Network Experiments. *ACM SIGCOMM Computer*

Communication Review, 54(1), 73-86. https://doi.org/10.1145/3618160

Appendix C – Deployment & Configuration Notes

1. ARO Cluster Provisioning

```
az provider register -n Microsoft.RedHatOpenShift
az group create -n aro-rg -l eastus2
az network vnet create -g aro-rg -n aro-vnet --address-prefix 10.0.0.0/16 \
--subnet-name master-subnet --subnet-prefix 10.0.0.0/23 \
--subnet-name worker-subnet --subnet-prefix 10.0.2.0/23
az aro create -g aro-rg -n promptcraft-cluster \
--vnet aro-vnet --master-subnet master-subnet \
--worker-subnet worker-subnet --location eastus2 \
--cluster-resource-group aro-cluster-rg --pull-secret @pull-secret.json
```

2. Helm Deployment

```
helm upgrade --install promptcraft ./infra -n promptcraft --create-namespace
```

- PodDisruptionBudgets, HPA, and NetworkPolicies are included.
- Secrets store API keys and Neo4j credentials securely.

Appendix D – Additional Tables

Table D1 – Experimental Cluster Specifications

Component	Specification
Cluster	ARO 4.17.27, 3 control-plane nodes (64 GB RAM / 16 vCPU each), worker pool auto-scaled 0–250 nodes (4 vCPU / 8 GB RAM per node)
LLMs	Phi-3 Mini (localhost), DeepSeek Coder (cloud), Mistral-7B (Azure AI)
Workloads	Manufacturing-IoT, Healthcare-FHIR, Generic-Code-Assist; 100 concurrent users issuing 5 prompts each
Metrics	Prompt latency, success-rate, engineering time, resource utilization