University of Amsterdam

ciena

DESIRE6G

FUTURE 6G NETWORK SERVICES

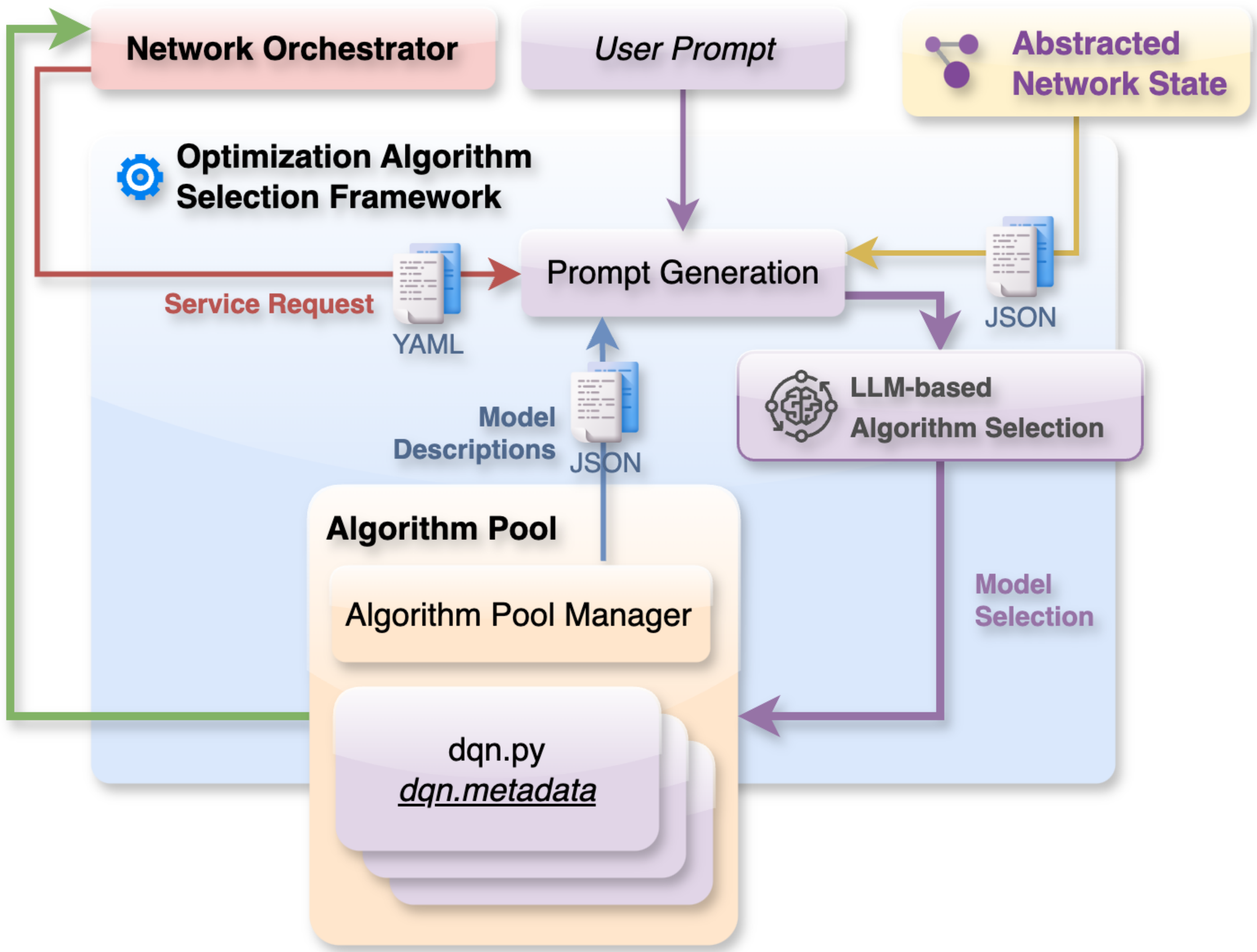
- The adoption of general **AI and LLM-driven workloads in every tool**, has resulted in an **exponential growth** explosion in terms of computational and networking demands across distributed infrastructures.
- Current infrastructure optimization **can not scale fast enough** to keep pace with demand, leaving significant resources under-utilized.
- Existing orchestration processes still **depend on manual decision-making to determine which KPI or objective to optimize**, resulting in static and time-consuming workflows.
- No single algorithm performs optimally across all scenarios.
- Clear need for a **context-aware, automated algorithm selection** mechanism.

- Instead of relying on a pre-set optimization algorithm or entirely offloading the optimization process to an LLM, our modular framework introduces a **hybrid decision layer**.
- It uses the **reasoning and contextual understanding capabilities of LLMs** to evaluate the current network state, service objectives, and available optimization methods.
- The LLM acts as an intelligent orchestrator, dynamically selecting the most suitable optimization algorithm from a curated pool, **emulating human expert decision-making, but with the scalability and speed required for real-time operation.**

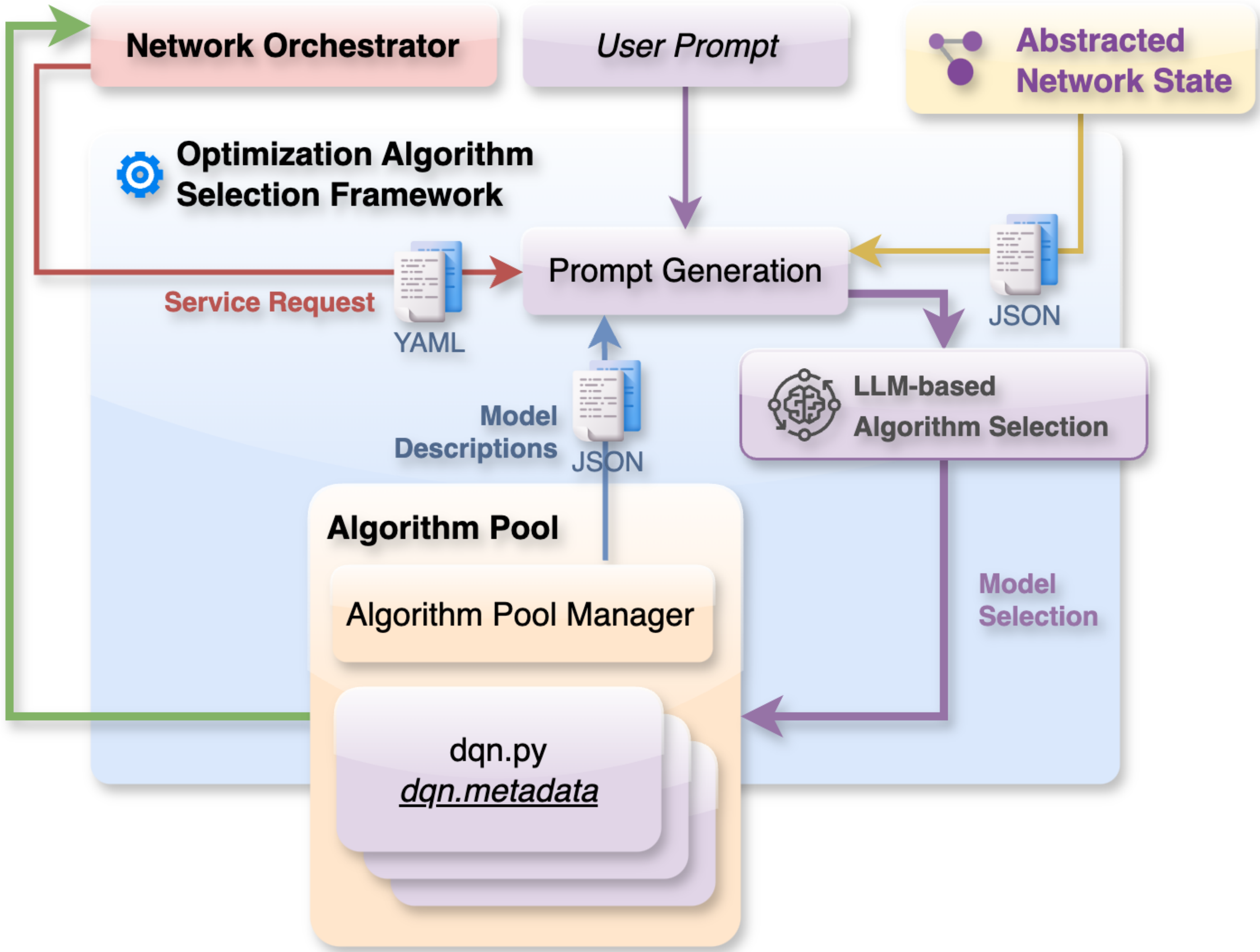
- The Algorithm Selection Problem was first formalized by John R. Rice (1976), quoting: *No single algorithm performs optimally across all problem instances*.
- Traditional methods rely on:
    - **Ranking and scoring systems** to evaluate candidate algorithms with weights.
    - **Feature extraction and clustering** of problem instances to guide selection.
- Modern approaches reimagine this concept with LLMs:
    - For **semantic feature extraction** and **context reasoning** directly from textual and numerical descriptions.
    - Combining **algorithm embeddings** and **problem embeddings** to identify the most suitable optimization strategy for the given context.
- Related work regarding Algorithm Selection exist mainly outside the networking domain. With our project we are bridging that gap, working on LLM-based algorithm selection for network orchestration.

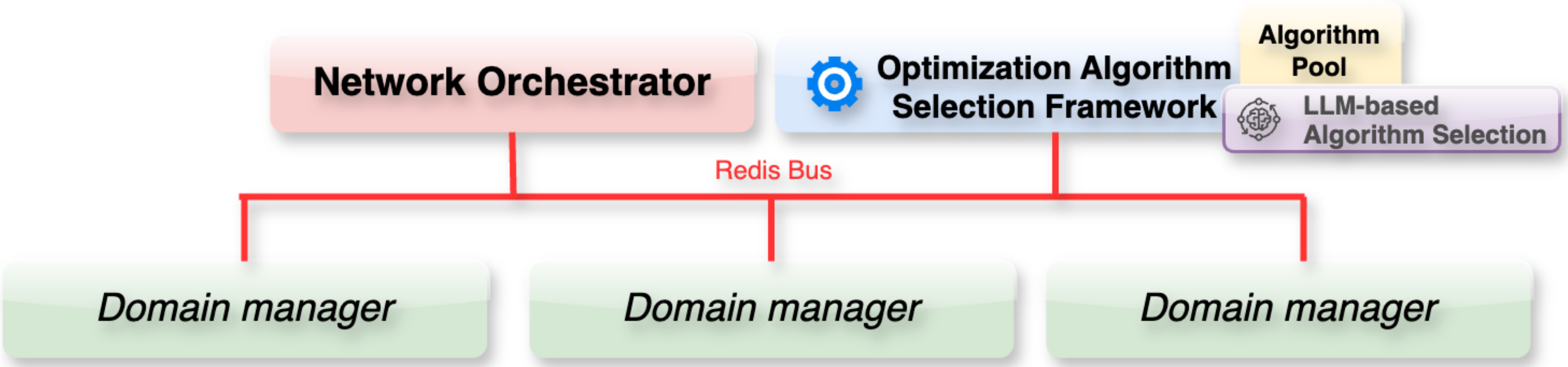
- Service Partitioning refers to the **process of dividing network services across multiple domains** to enhance performance and resource efficiency.
- The optimization objective that we explore is to perform Algorithm Selection to optimize several KPIs, for example: latency.
- Common approaches in the literature:
  - **Classical methods:** Integer Linear Programming (ILP), heuristic optimization
  - **Machine Learning methods:** Reinforcement Learning (RL), Double Deep Q-Networks (DDQN), and Multi-Agent RL for decentralized decision-making.
- Service partitioning is selected as the preliminary use case for evaluating the proposed LLM-based algorithm selection framework, it provides a complex, multi-objective orchestration problem representation of real-world network operations.

- We have designed a modular LLM-based **Algorithm Selection Framework**.
- The **LLM interprets text-based information** such as operational logs, service descriptions, and network state metrics.
- **Dynamically selects** the most suitable optimization algorithm from a curated pool of optimization strategies.
- It acts as an **abstraction layer** one step before the optimization process.

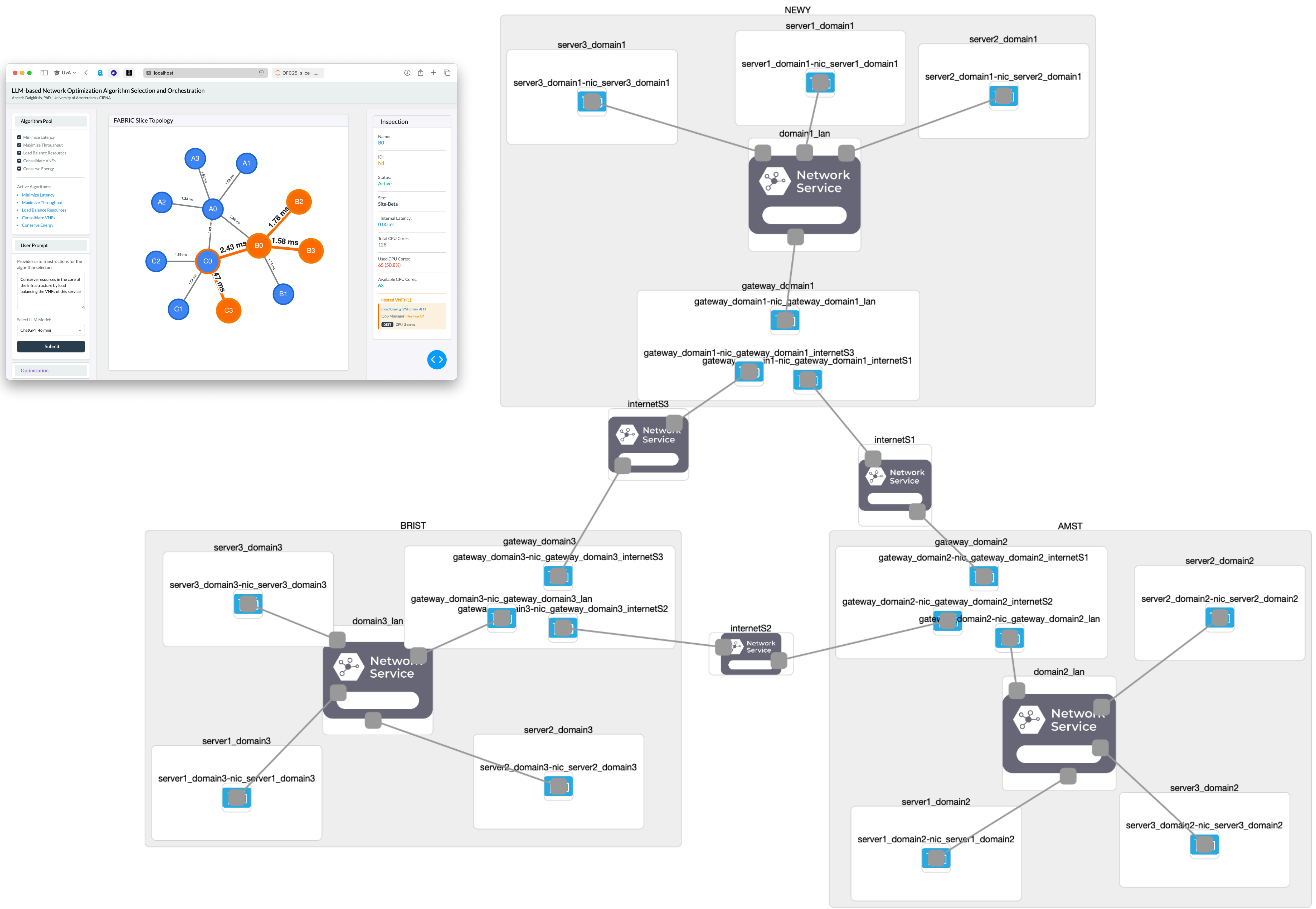


- **Algorithm Pool:** Repository of optimization algorithms and their metadata (AST, complexity, performance).
- **Prompt Generator:** Converts service and network state logs into LLM-ready prompts, including custom user instructions.
- **LLM Selector:** Inference and selection of the the optimal algorithm choice according to the prompt instructions.
- **Interface:** API to connect with other orchestration systems in real time.



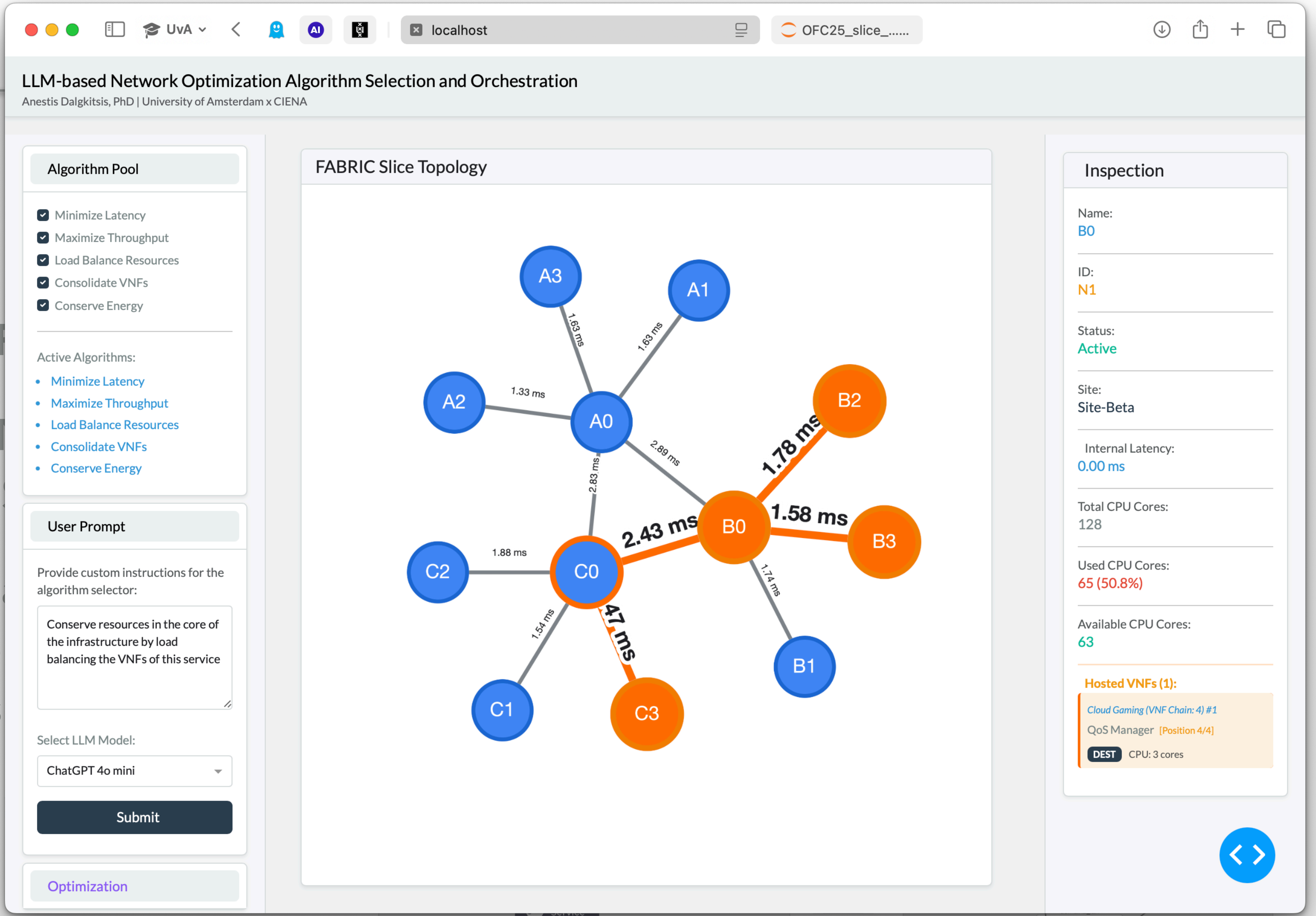


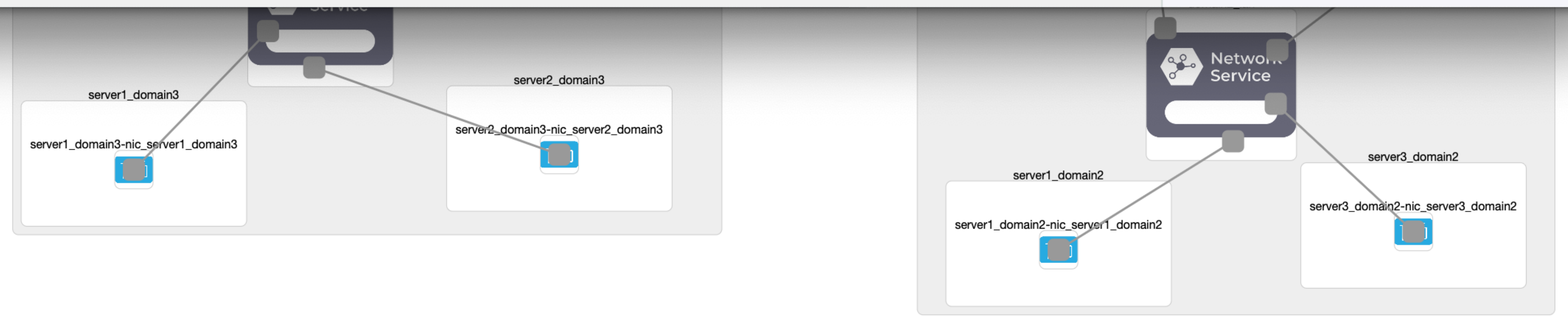
- **Testbed:** Demo deployed on **FABRIC**:
  - **Multi-domain** slice: New York (NEWY), Amsterdam (AMST), Bristol (BRIST).
- **Implementation:** Python, Redis (databases, messaging bus), Docker (Container VNFs), net tools (performance measurements).
- **Models:** GPT-4o-mini (3rd party) and Llama 3.2 1B (Ollama local).
- **Dashboard:** Real-time, interactive visualization playground to explore the concept.



SUPERCOMPUTING 2025 LIVE DEMO

- **Testbed:** Demo deployed on **FABR**
  - **Multi-domain** slice: New York (N
    Amsterdam (AMST), Bristol (BRIS
- **Implementation:** Python, Redis (d
  messaging bus), Docker (Containe
  tools (performance measurements
- **Models:** GPT-4o-mini (3rd party) a
  3.2 1B (Ollama local).
- **Dashboard:** Real-time, interactive visualization playground to explore the concept.



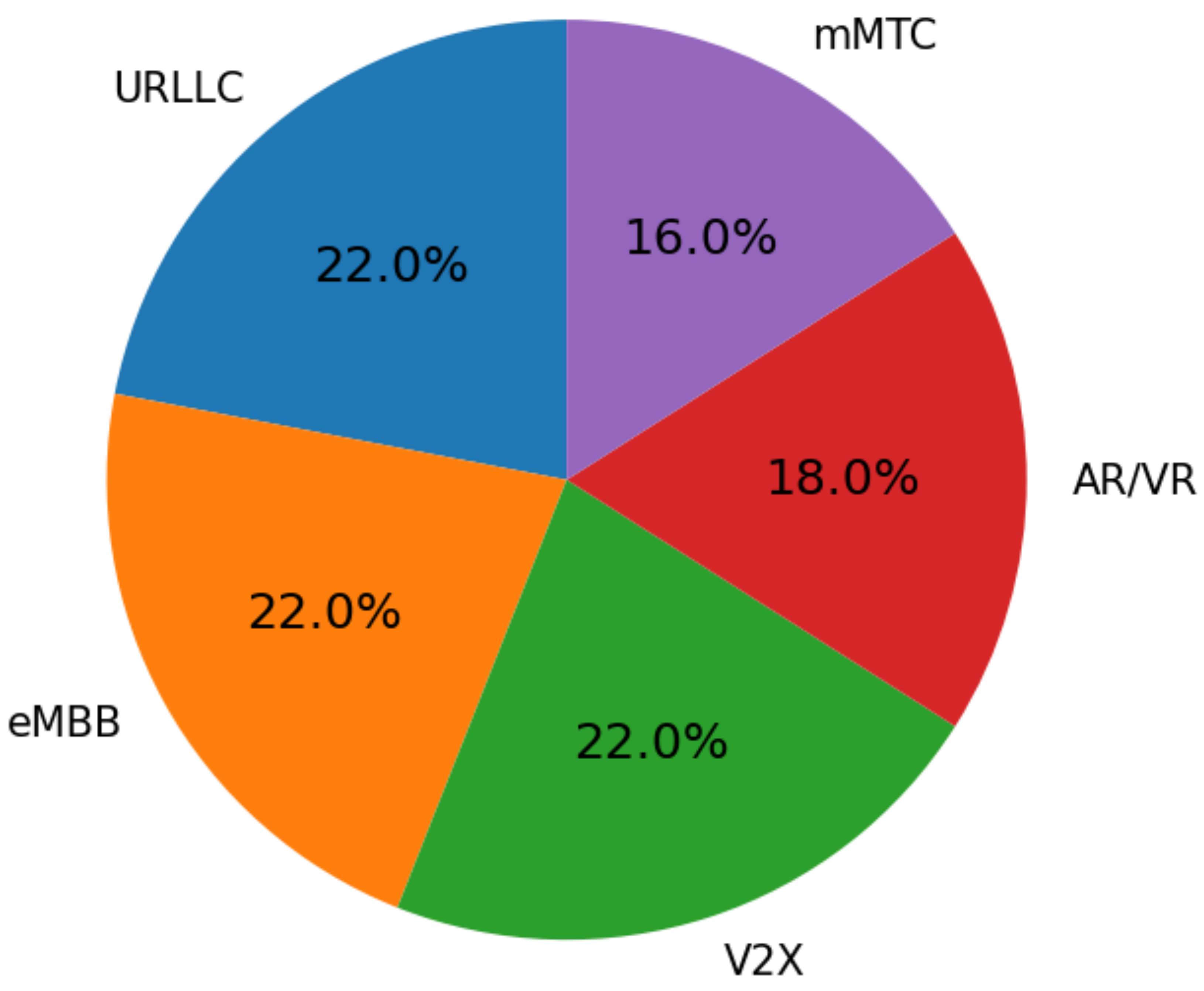


- **Service Scenarios:** Evaluation is conducted using common 5G/B5G/6G service profiles, representing diverse network requirements:
  - eMBB, URLLC, mMTC, V2X, and AR/VR.
- **Performance Metrics:** Two primary indicators are used to assess framework performance with simulation:
  - **Partitioning Success Rate:** Percentage of successfully deployed service partitions.
  - **SLA Compliance Rate:** Admitted services that fully meet SLA requirements.
- **Success conditions:** A partition is considered successful when:
  - **Resource constraints** are met (over provisioning is allowed with performance degradation).
  - **Valid VNF placement** across domains.
  - **SLA Latency requirements** are met.

Table 1: Service Types

Service Type	λ (ms)	τ (Mbps)	Reliability
eMBB	1–10	100–1000	99.9%
URLLC	0.5–5	10–100	99.999%
mMTC	10–100	1–10	99%
V2X	1–5	50–200	99.99%
AR/VR	5–20	200–500	99.9%

Service Type Distribution



Partitioning Methods (Pool)

- DDQN RL (single agent)
- DDQN MARL (multi-agent)
- Greedy
- SLA-aware
- Logic-based
- Resource-based (CPU)
- Random

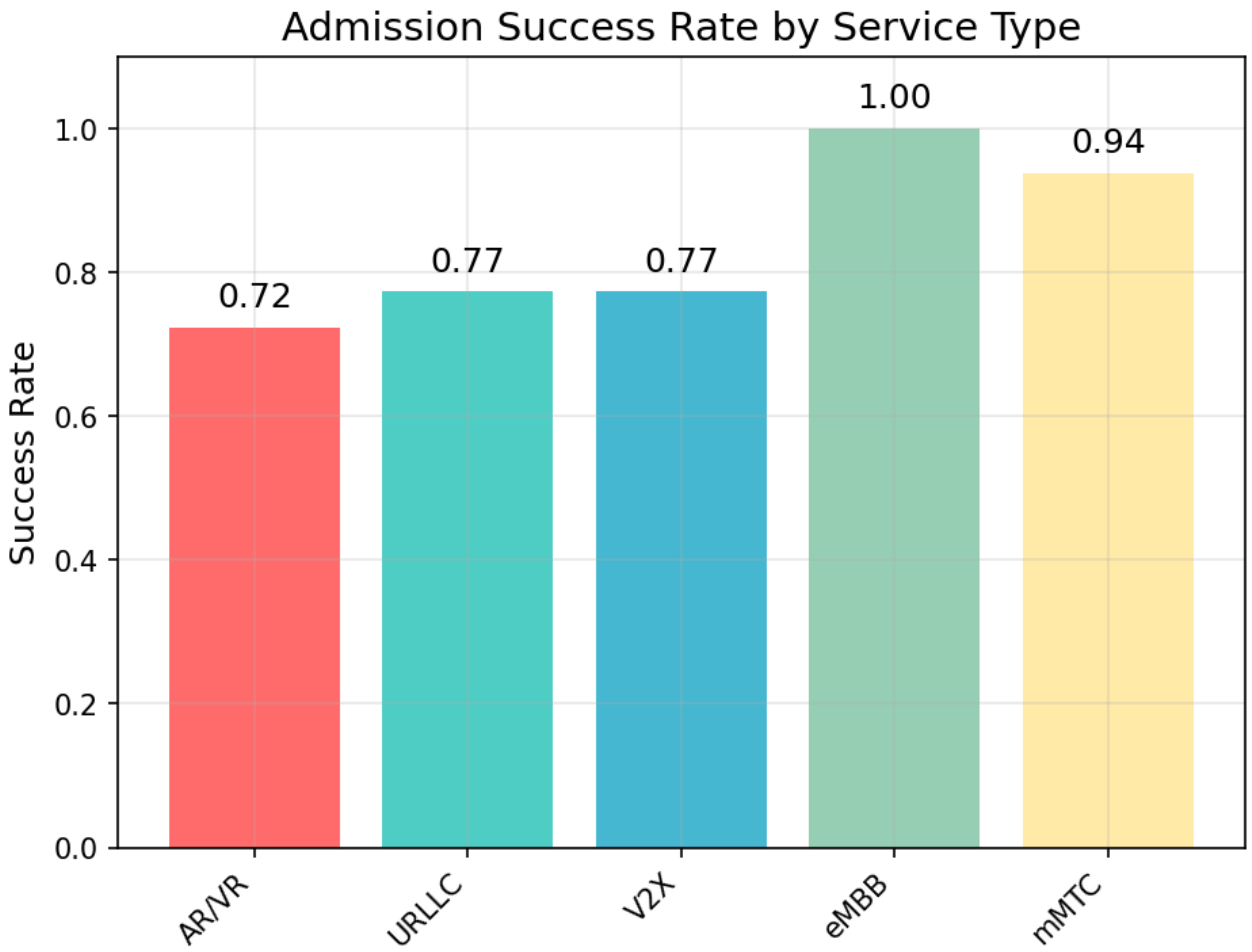
LLM Models Compared

- Llama 3.2: 1b (local)
- Llama 3.2: 3b (local)
- GPT-4o (3rd party)
- GPT-4o-mini (3rd party)

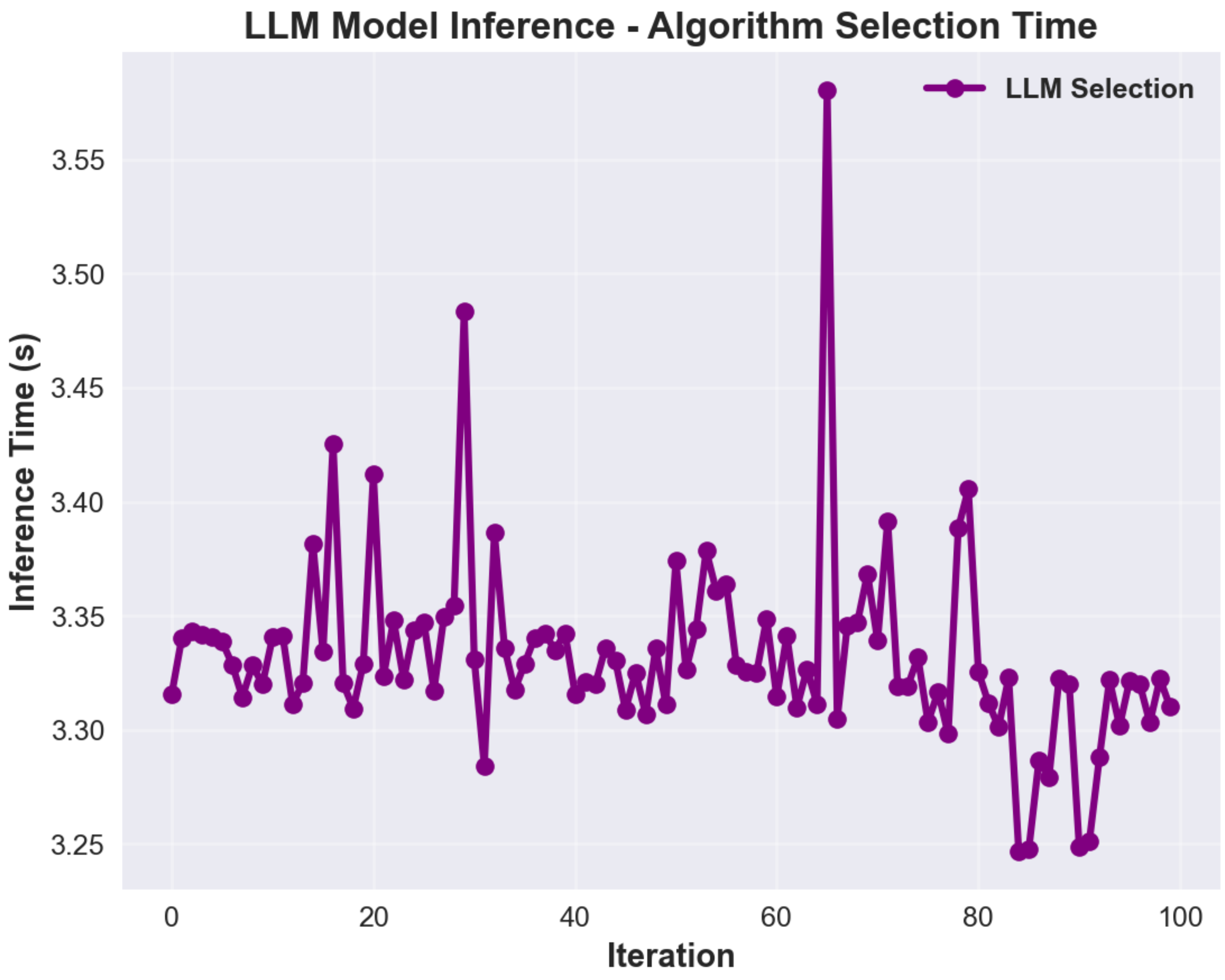
- **100 iterations** using Llama 3.2:3B.
- AR/VR, URLLC, and V2X hardest to admit due to strict latency limits.
- eMBB, mMTC maintain success due to relaxed latency SLA.

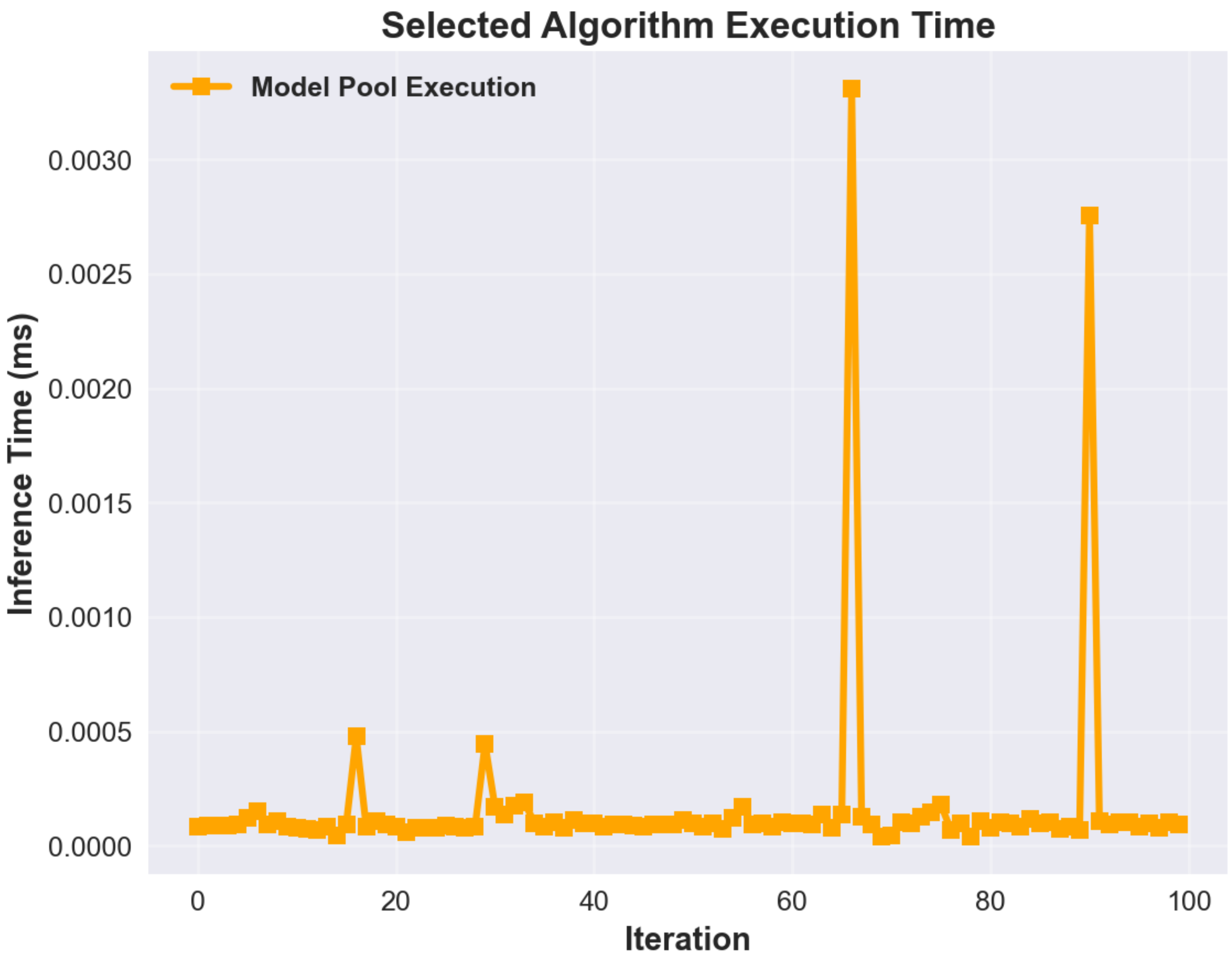
Table 1: Service Types

Service Type	λ (ms)	τ (Mbps)	Reliability
eMBB	1–10	100–1000	99.9%
URLLC	0.5–5	10–100	99.999%
mMTC	10–100	1–10	99%
V2X	1–5	50–200	99.99%
AR/VR	5–20	200–500	99.9%

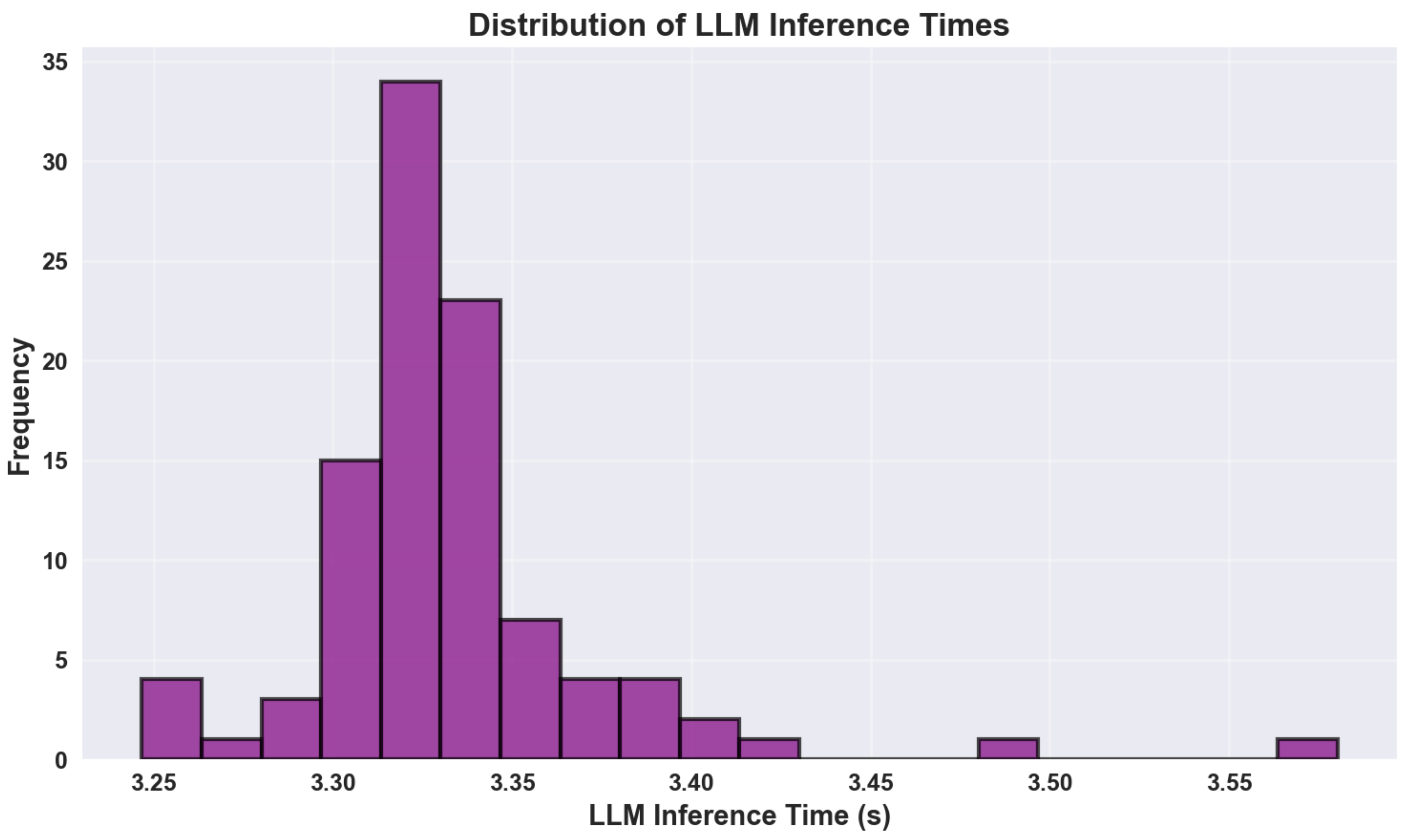


- LLM inference time remains stable across iterations (~3.3s avg)
- Selection alternates mainly between Greedy and Single DDQN solutions
- Occasional spikes when prompt complexity increases due to resource scarcity.
- Local LLM models show low and consistent overhead suitable for real-time orchestration. Data privacy too.



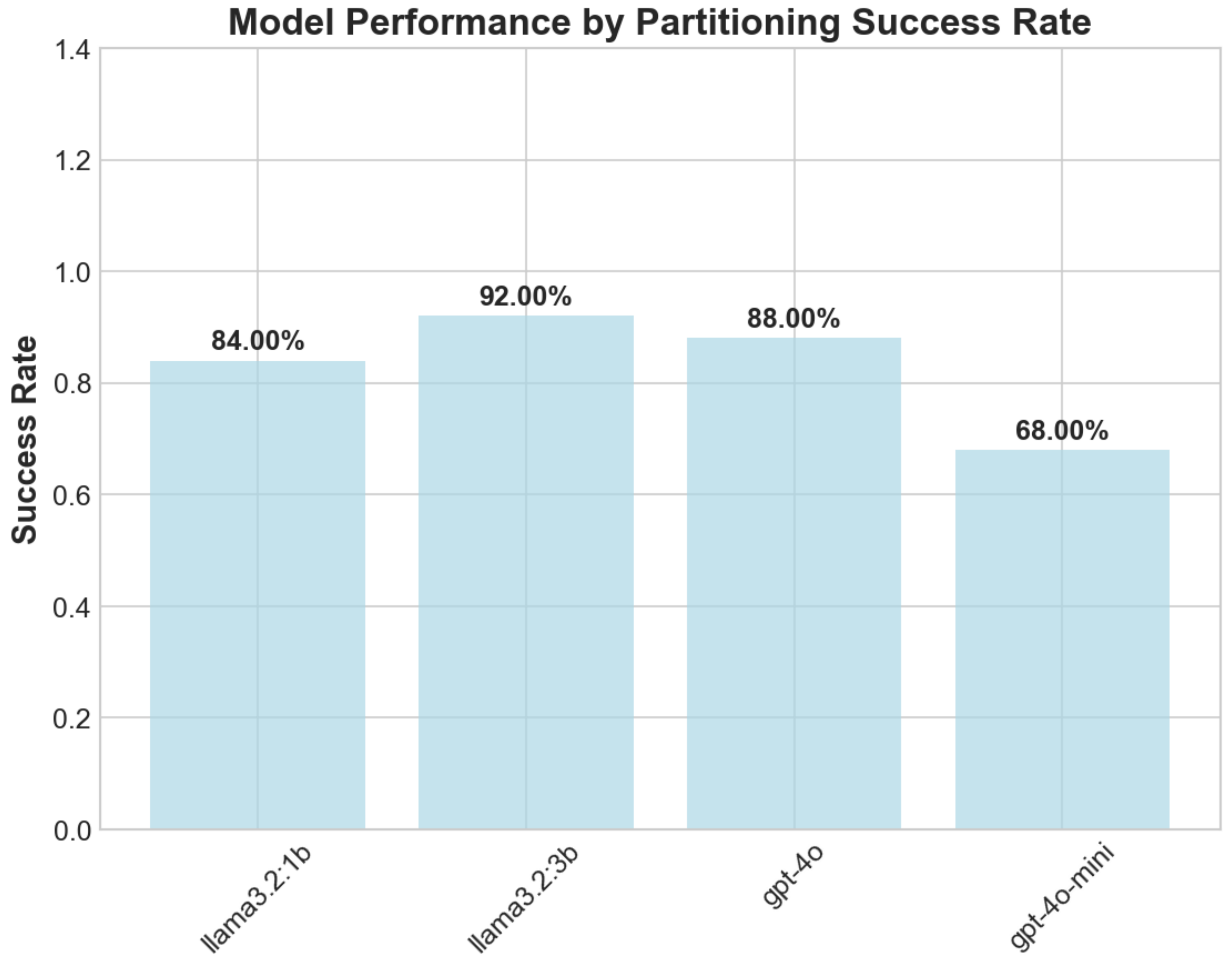


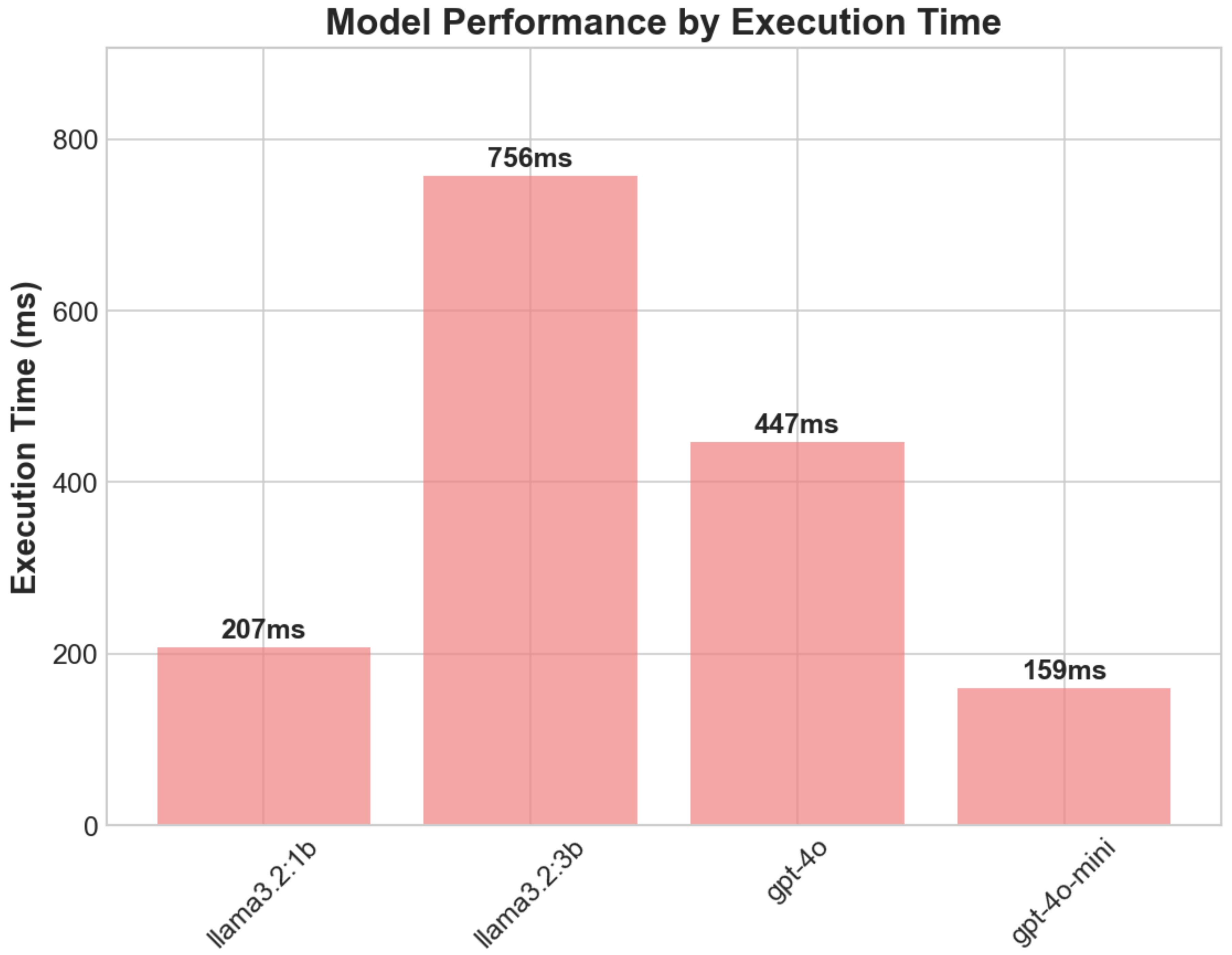
- Stable inference distribution centered around ~3.3s.
- Predictable inference latency for local inference (under normal compute load).
- Local, on premise LLMs provide consistent timing for closed-loop orchestration.



- Llama 3.2:1B (local) performs surprisingly close to larger models (due to simple preliminary study scenario).
- **Llama 3.2:3B** achieves highest **success rate (≈92%)**, optimal balance of accuracy and speed.
- Key takeaway: **Smaller LLMs can be viable** (for simple scenarios).

- GPT-4o yields strong accuracy (≈88%) but higher latency due to *"Thinking"* and external inference.
- GPT-4o-mini fastest with external inference (≈160ms) but least consistent (~68 %).
- Key takeaway: There are **trade-off between performance and inference cost**.





- Llama 3B: highest success, moderate latency overhead (local inference), data privacy.
- GPT-4o: Costly inference, inference latency.
- Llama 1B: Lightweight and stable, runs on limited resources, data privacy.
- GPT-4o-mini: Fast response, costly inference.
- **Preliminary results:** Confirm that **mid-sized local ≈3B models can yield viable cost-performance ratio** for optimization algorithm selection.

Model	**Success %**	λ **(ms)**	τ **(Mbps)**	**Time (ms)**
llama3.2:1b	84.18%	15.45	176.29	207.360
llama3.2:3b	92.15%	15.85	178.14	756.445
gpt-4o	88.32%	14.87	208.06	447.103
gpt-4o-mini	68.45%	15.08	209.85	158.993