Journal of Engineering Managment and Soft Computing (JEMSC)

Co-EvoDT: Design of a Co-evolving Digital Twin Framework for Media Digital Transformation in the Metaverse through the Integration of Multi-Agent Reinforcement Learning and LSTM-based Sequential Forecasting

Seyedeh Tahereh Mousavi[1], Farshad Faezy Razi[2]✉ and Abolfazl Danaei[3]

1. PhD Student in Media Management, Department of Media Management, Se.C., Islamic Azad University, Semnan, Iran. Email: seyedehtahereh.mousavi@iau.ac.ir
2. Corresponding Author. Associate Professor, Department of Industrial Management, Se.C., Islamic Azad University, Semnan, Iran. Email: fa.faezy@iau.ac.ir
3. Associate professor, Department of Media Management, Se.C., Islamic Azad University, Semnan, Iran. Email: ab.danaei@iau.ac.ir

Article Info

Article type:
Research Article

Article history:
Received 30 September 2025
Received in revised form
20 April 2026
Accepted 13 June 2026
Published online 1 July 2026

Keywords:
media digital transformation, metaverse, digital twin, multi-agent reinforcement learning, sequential forecasting, quality of experience (QoE).

ABSTRACT

Digital twins in the metaverse have created new opportunities to redefine digital transformation in media; however, the absence of co-evolutionary models that simultaneously optimize infrastructure resource-allocation policies, content production, and load forecasting remains a major barrier to fully realizing these opportunities. To address this gap, we propose the Co-EvoDT model and implement it within a simulated metaverse environment using a design-driven, quantitative research approach. The proposed architecture combines sequential load forecasting via an LSTM predictor (prediction horizon = 1, sequence length = 8), multi-agent reinforcement learning trained with the policy-based REINFORCE algorithm to enable co-evolutionary training of content, infrastructure, and user digital twins, and a runtime rolling-update mechanism that appends the LSTM's normalized output to the state vector used by the control policies. Performance was evaluated using Quality of Experience (QoE), latency, cost, and RMSE of the load predictor. Simulation results show that a policy trained with the combined predictive signal outperforms a random policy by simultaneously improving multiple metrics: a relative increase of ≈93% in QoE, a relative latency reduction of ≈29%, and a cost reduction of ≈27%. Forecast RMSE was reduced by ≈43%, ≈53.2%, and ≈88% compared to Naive, ARIMA, and Exponential Smoothing baselines, respectively. Reward-curve convergence analysis and parametric sensitivity experiments further corroborate the stability and robustness of the learned policies under variations of key system parameters. The principal innovation of this research is the operational integration of multi-agent reinforcement learning, sequential forecasting, and co-evolutionary population-gradient update mechanisms within digital twins to enable proactive, coordinated resource management in the metaverse. The results indicate that Co-EvoDT can concurrently enhance user experience and macro-level system performance.

Cite this article: Mousavi, S.T. & et al, (2026)., Co-EvoDT: Design of a Co-evolving Digital Twin Framework for Media Digital Transformation in the Metaverse through the Integration of Multi-Agent Reinforcement Learning and LSTM-based Sequential Forecasting. *Journal of Engineering Management and Soft Computing,* 12 (3). 39-61.

DOI: https://doi.org/10.22091/jemsc.2026.14030.1306

DOI: https://doi.org/10.22091/jemsc.2026.14030.1306

Publisher: University of Qom

1) Introduction

The convergence of novel technologies in the metaverse ecosystem emphasizes the necessity of dynamizing adaptive modeling to predict and manage, in real time, the technical, content, and behavioral components. In this perspective, "co-evolving digital twins"—aimed at learning and representing the dynamics of complex media ecosystems—organize their operation across two domains: simulating the real-time state of media infrastructure/content to recreate environmental conditions, and then interacting with autonomous decision-making agents to increase responsiveness. Thus, the intervention of digital twins in media digital transformation encompasses, in addition to guaranteeing the quality of the user experience, reductions in latency and network load costs, as well as the optimal allocation of infrastructural resources.

The synergy of these intelligent agents alongside one another requires an operational framework in distributed systems—a concept investigated in recent years under the rubric of learning in multi-agent environments (MARL). Based on findings, digital twins for reproducing the evolution of the physical and social world require continuous, even supra-temporal, reconstruction and updating. In this vein, studies that delineate the infrastructural role and simulation of digital twins within metaverse architectures have emphasized the necessity of employing predictive and control models. From the temporal-forecasting perspective, the applications of deep neural LSTM networks in improving load handling and reducing latency in distributed systems have also been addressed. Moreover, studies highlight that employing a multi-agent reinforcement learning (MARL) approach for scheduling and resource allocation in cloud environments can reduce costs.

Despite progress, important gaps persist in the theoretical literature. First, the absence of an integrated co-evolutionary model for digital twins in the metaverse is evident—a model that can simultaneously optimize policies for producing interactive content, forecast, and improve network load. Second, there is a lack of demonstrated effectiveness of integrating LSTM-based load forecasting with a multi-agent reinforcement learning (MARL) algorithm based on REINFORCE in the metaverse for the simultaneous improvement of learning stability, quality of experience, and cost.
Third, the absence of efficient reward mechanisms and data architectures aimed at ensuring convergence and transferability under high uncertainty is another challenge to enhancing digital twin performance in metaverse ecosystems.

Therefore, to cover the aforementioned gaps, this research proposes a co-evolutionary and simulated framework named Co-EvoDT that combines multi-agent reinforcement learning based on policy-gradient updates (the REINFORCE algorithm), a population co-evolutionary process over a set of policy networks, and an LSTM-based sequential load predictor. This co-evolutionary framework is based on three factors: user, content, and infrastructure. Although "co-evolutionary" does not mean simple, simultaneous learning of multiple agents in a shared environment, here "co-evolutionary" denotes a bidirectional combination of online parameter updates based on the policy gradient (the REINFORCE method for rapid correction of a sampled member from each population) and periodic population evolutionary steps (tournament selection, member evaluation, parameter aggregation via averaging and Gaussian mutation on the weights, replacement of weak members). In this framework, evolution is confined solely to the weights of the policy networks, while network architectures and high-level meta-hyperparameters are kept constant; the output of the memory-based sequential predictor is injected as an additional feature into the state vector and is periodically fine-tuned online. This gradient–population overlap enables the retention and exploration of multimodal behavioral diversity, combining long-term evolutionary exploration with short-term gradient adaptation, reducing the risk of mode collapse and enhancing resilience when facing environmental changes. This offers a novel perspective for designing stable and environment-adaptive digital twins suitable for variable and complex metaverse environments; empirically, this combination leads to noticeable improvements in latency and cost metrics and enhances the quality of experience compared to classical MARL methods. This constitutes the primary innovation of the method in integrating evolution-driven exploratory capabilities with rapid gradient corrections and leveraging sequential forecast signals.

The remainder of this paper is organized as follows: Section 2 reviews conceptual literature, prior studies, and foundational theories. Section 3 details the simulation methodology, algorithm implementation, and the proposed architecture. Section 4 presents experimental findings and analyzes the results. Section 5 discusses the practical implications of the research findings, concludes, and offers suggestions for future work.

2) Theoretical Background

2-1) Theoretical Foundations

2-1-1) Metaverse

The metaverse refers to virtual reality that goes beyond ordinary reality. The term denotes the digitalized terrain as a new world that is expressed through digital media such as smartphones and the Internet (Rezaeenour & Karimian, 2024; Rezaeenour & Karimian, 2025). In other words, the metaverse is an ecosystem for connecting virtual and physical environments to one another and can be altered and modeled from within by multiple potentially immersed users. It is a persistent and enduring three-dimensional world implemented through virtual, augmented, and physical reality. The metaverse is entirely about communication: between the user and the virtual world, between the user and the physical world, between the physical and virtual worlds, and between different virtual worlds (Benaben et al., 2025; Chow et al., 2022).

2-1-2) Digital Twin

A digital twin is a live virtual replica of a physical twin, such as a product, asset, component, process, or system. This replica is formed with the support of technologies such as artificial intelligence, machine learning, sensors, and the Internet of Things (IoT). A data flow is established between the digital twin and the physical twin to ensure that digital twins are continuously updated and adapt to changes in their physical counterparts. This enables the sharing of insights, decision support, and provides the capability to simulate, predict, monitor, control, and optimize the performance of the physical twin across its lifecycle. Digital twins can range from simple instances to complex models with self-learning and adaptive capabilities, depending on their level of complexity (AlBalkhy et al., 2024; Alvi et al., 2025).

2-1-3) Media Digital Transformation

It is defined as a transformative, paradigmatic, and evolution-driven process that involves deep and continuous redesign and re-engineering of media workflows, structures, processes, and methodologies through integration with key drivers (digital technologies). This transformation is fundamentally deeper than mere digitization and represents a rethinking and operational redefinition of the processes that govern how media content is created, distributed, and consumed (Mousavi et al., 2025; Novikov et al., 2023)

2-2) Theoretical Backing

2-2-1) Complex Systems Theory

Complex systems theory refers to the study of systems characterized by interdependent components that interact in complicated ways and can provide insights into why certain practices succeed or fail in dynamic environments. By revealing the limitations of mechanistic approaches and asserting that adaptation with incomplete information is essential for effective operation in such systems, this theory challenges traditional methods. Given the dynamics and nonlinear changes of the metaverse ecosystem, the use of digital twins for testing changes and updating them is justified (Haimes, 2018).

2-2-2) Co-Evolutionary Theory (Concurrent Development)

Co-evolutionary theory describes reciprocal evolutionary changes between species or interacting systems, in which the evolution of each entity influences and is influenced by the evolutionary path of the other. The co-evolution metaphor is used to conceptualize the mutual relationships among

technological innovations, human behaviors, and institutions. Actors influence one another directly or indirectly through social institutions. In a co-evolutionary process, the adaptive landscape of one actor changes and is transformed as other actors make their adaptive moves. Each party in the interaction plays a role in the development of the other by exerting selective pressures and mutually adapting (Rammel et al., 2007). Considering that multiple digital agents (as digital twins) interact with one another and with the real environment in the proposed model, a co-evolutionary framework was adopted.

2-2-3) Multi-Agent Reinforcement Learning (MARL)

Multi-Agent Reinforcement Learning (MARL) builds on reinforcement learning (RL) and deep reinforcement learning (DRL) and extends the Markov Decision Process (MDP) decision-making framework to environments with multiple agents. In MARL, agents interact both with the environment and with each other, and each follows its own policy (policy). In other words, multiple agents are trained and interact in shared game-like environments and learn policies via reinforcement learning to improve their performance through trial and error (Hady et al., 2025; Li et al., 2025; Salamattalab et al., 2024).

2-2-4) Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks were developed to address this problem. LSTMs are a type of recurrent neural network (RNN) and are particularly effective at learning long-term dependencies. LSTM cells have a special structure that allows information to be stored for longer periods without being lost. This cell structure includes components such as a forget gate, an input gate, and an output gate. While the forget gate determines which information to forget, the input gate controls how new information is added to the cell. The output gate specifies which information from the cell state is transferred to the new hidden state. Using these gates and the cell state, LSTMs effectively learn long-term dependencies in time-series data and perform forecasting (Olcay et al., 2024)

2-3) Critical Literature Review

Given the absence of co-evolutionary digital-twin algorithms for the metaverse in the scholarly literature, we analyzed studies related to the evolution of digital twins: Bersani et al. (2022) proposed an initial framework for digital-twin architecture aimed at ensuring that a digital twin matches its corresponding physical twin (PT) in configuration and behavior; this framework includes appropriate modeling primitives, trust assurance, and co-evolution between digital twins and the physical specimen (Bersani et al., 2022). Tank et al. (2024), by introducing “co-evolving digital twins” (CoEDT), created a parallel world that maps co-evolutionary behavior patterns for DTs to co-evolutionarily optimize multiple digital twins across product lifecycles using bio-cellular methods (Tong et al., 2024). Feng et al. (2025) proposed a co-evolutionary approach for safe motion planning in human–robot industrial interaction using a human digital twin (HDT) that supports mutual and continuous evolution of human safety operational cognition and robot safe-motion planning strategies (Feng et al., 2025). Fan et al. (2025) presented a hybrid DT model combining physical systems and “soft” systems (human behavior, decision-making) to improve green production. Kang et al. (2025) proposed a novel adaptive evolution approach for digital twins by combining physics-informed DeepONets and transfer learning, which, even with limited data, could detect changes in physical systems and produce accurate long-term forecasts (Cong et al., 2025). Tang (2024) put forward a co-evolving digital-twins (CoEDTs) framework inspired by cellular biology and the integration of digital threads and MBSE for embedding digital twins in the product lifecycle. Despite the development of co-evolutionary applications of digital twins in the cited research, none have simultaneously examined the combination of MARL (REINFORCE) and sequential forecasting (LSTM) for the multi-objective optimization of user experience, latency, and cost in dynamic, real-time metaverse scenarios; therefore, the present study fills this gap by introducing Co-EvoDT and empirically demonstrating the performance and stability of the proposed approach through quantitative simulations.

2-4) Conceptual Framework

The proposed algorithm, grounded in theoretical supports, implements the interaction and co-evolution of three main factors—content (type and volume of media produced), infrastructure (communication and computational networks), and user (consumption patterns and feedback)—within a metaverse simulation environment based on digital twins. Each of these factors is modeled by a dedicated digital twin whose policies and behaviors affect one another; for example, a shift in user preferences increases content production patterns and graphical processing and network traffic load, which in turn indirectly impacts the mapping/allocation of infrastructure resources.

In this complex model, to improve synergistic and synchronized decision-making (e.g., bandwidth allocation, content format changes, or personalization of experience) among independent agents (the three digital twins), two key AI tools are employed: a deep LSTM model that forecasts future demand based on historical network-load and user-behavior data. This load forecast enables the digital twins to simulate the necessary resources and pre-position responses to instantaneous changes. The LSTM output, after normalization, is appended to the existing state to be used as an input feature in the agents' decision processes.

Additionally, a set of multi-agent reinforcement learning (MARL) agents representing digital twins are trained concurrently in a shared framework to learn optimal resource-allocation and content-management policies. These agents continuously update their policies based on specified rewards (such as user satisfaction and resource-use efficiency). In this way, digital twins are co-evolutionarily optimized through mutual interaction with users and infrastructure.

Finally, in this dynamic framework, the produced content, user actions, and infrastructure status are modeled simultaneously, and autonomous optimization processes are executed using sequential LSTM forecasting and collaborative MARL learning. The reason for using policy-based REINFORCE is the implementation of multifaceted reward functions for agents, and LSTM is used for load forecasting based on its ability to learn long-term sequential patterns. The LSTM output is a point forecast with a horizon (h) equal to one. This forecast, after min–max normalization to the [0,1] interval, is concatenated to the state vector:

$$s_{in} = \mathrm{concat}\left(s, \hat{\ell}_{t+1}\right).$$

where $\hat{\ell}_{t+1}$ is the predicted value at time t+1 (after normalization).

3) Research Method

This applied and developmental study, using a design-science research (DSR) methodology and a quantitative approach, performs a structured evaluation via digital-twin simulation.

3-1) Digital-Twin Simulation

Within the DSR cycle, an interactive digital twin based on multi-agent reinforcement learning (MARL) with the REINFORCE algorithm and an LSTM predictor was implemented. The code was executed on a server with an 8-core CPU, 32 GB of memory, and an NVIDIA RTX 3080 GPU with GPU (PyTorch) support, using Python. The simulation was designed and the models were trained using Python 3.9 and the libraries NumPy, Pandas, and PyTorch for implementing the LSTM, reinforcement learning, and data processing. Execution steps include baseline model generation, LSTM pre-training, MARL training using state augmentation, and—in online mode—performing a rolling update after every p episodes.

3-2) Assumptions

• The CoEvoEnv simulation environment is a simplified metaverse model of interaction among three agents/digital twins (content, infrastructure, user).

• Updates to dynamic variables are performed via small random noises to produce stochastic outcomes.

• The environment state consists of a five-dimensional vector with arrays for virality, load, infrastructure capacity, content quality, and the time-to-episode-length ratio.

• Each agent affects at each time step one of three discrete actions {0,1,2}, corresponding respectively to content promotion/dissemination, allocation of infrastructure resources, and the level of user participation.
• Each agent has a separate reward function determined by weighting performance metrics such as QoE, cost, and latency.

• This model is intended solely to study co-evolutionary behavior and MARL policies at an experimental scale.
• Variables are initially randomized with a fixed SEED (SEED = 42) to provide baseline reproducibility. In each independent run, the seed is set according to the relation seed_run = BASE_SEED + run to preserve statistical variety.

• At the start of each episode, the initial infrastructure capacity is assumed to be fixed and equal to 0.5 (c = 0) to provide the same starting point for all simulation scenarios.

• For online LSTM updates, the parameter p = LSTM_UPDATE_FREQ_EPISODES (online LSTM update frequency, unit: episodes) is used; i.e., updates occur only in episodes whose indices are multiples of p.

• Evaluation metrics include mean cost, latency, quality of experience (QoE), and RMSE for load forecasting.

• The parameter K (EPISODE_LEN) denotes the length of each simulation episode in time steps and is set to 50. The choice of 50 stems from preliminary experiments to strike an optimal balance between providing a sufficiently long horizon for learning long-term policies and controlling computational complexity and preventing instability in the training process. This parameter not only determines the decision-horizon length for agents but also directly impacts REINFORCE learning (via episodic discounted-return computation) and the volume of data available for pre-training and updating the LSTM model.

3-3) Digital Twin–Metaverse Simulation Algorithm

Algorithm: MARL (REINFORCE) + population co-evolution + LSTM predictor

Inputs / hyperparameters:

MARL_EPISODES (e.g. 200)
EPISODE_LEN = K (e.g. 50)
BASELINE_TRACE_EPISODES (e.g. 160)
EVAL_EPISODES (e.g. 80)
LSTM_EPOCHS (e.g. 20), LSTM_SEQ_LEN = 8, LSTM_HIDDEN = 32, LSTM_LR (e.g. 1e-3)
POP_SIZE (e.g. 8), EVOLVE_FREQ (e.g. 40), EVAL_PER_MEMBER (e.g. 2)
TOURNAMENT_K (e.g. 3), MUT_STD (e.g. 0.02)
ONLINE_UPDATE_FREQ = p (e.g. 10, unit: episodes)
POLICY_LR (e.g. 5e-3), $\gamma = 0.99$
MODE $\in$ {NoPred, PretrainOnly, Online}
N_RUNS (e.g. 10), SEED

Outputs:

- Trained agent policies (best per agent)
- Trained LSTM model (if MODE $\neq$ NoPred)
- Per-run and aggregated metrics: Avg_QoE, Avg_Latency, Avg_Cost (mean $\pm$ std)
- RMSE table for LSTM vs baselines (Naive, ExpSmooth; ARIMA)
- Convergence plots, population_scores, per-episode traces (CSV)

Procedure:

1. For run = 1 … N_RUNS:

1.1 Set random seeds ← SEED + run, set device (CPU/GPU).

1.2 Instantiate environment factory: Env ← CoEvoEnv(EPISODE_LEN =k).

1.3 Baseline trace collection:

for ep = 1 … BASELINE_TRACE_EPISODES:

s ← Env.reset()

while not done:

actions ← random actions for all agents

s', rewards, done, info ← Env.step(actions)

store per-step trace (state, actions, info)

Compute baseline per-episode lists and baseline mean±std for QoE, latency, cost.

1.4 Pre-train LSTM (if MODE ≠ NoPred):

- Build dataset of load sequences from baseline traces with seq_len = LSTM_SEQ_LEN.

- Train LSTM (architecture: LSTM(hidden=LSTM_HIDDEN) → FC → scalar) for LSTM_EPOCHS using MSE.

- Evaluate LSTM on a holdout and compute RMSEs for:

• LSTM

• Naive persistence: \hat{l}_{t+1}^{Naive} = l_t

• ARIMA fit (least-squares on last-value → b0 + b1 * last)

• Exponential smoothing (fixed α)

(If statsmodels available, optionally evaluate ARIMA) .

1.5 Initialize population of policy networks:

- If MODE == NoPred: policy_input_dim = 5 (env state)

else: policy_input_dim = 6 (env state + LSTM prediction)

- For each agent i ∈ {content, infra, user} create POP_SIZE PolicyNet(input_dim=policy_input_dim).

- For each member create its own Adam optimizer (lr=POLICY_LR).

1.6 MARL training loop (REINFORCE + periodic co-evolution):

for ep = 1 … MARL_EPISODES:

- Sample one member index per agent: chosen_idx[i] ~ Uniform(0, POP_SIZE).

- members[i] ← population[i][chosen_idx[i]]

- s ← Env.reset(); initialize recent_load deque (maxlen=LSTM_SEQ_LEN)

- while not done:

- if MODE == NoPred or LSTM not available:

s_in ← s

else:

if recent_loads has LSTM_SEQ_LEN:

ŷ ← LSTM.predict(recent_loads) # horizon = 1
else:
ŷ ← last observed load
s_in ← concatenate(s, ŷ) # state-augmentation
- For each agent i: sample action a_i ~ π_{θ_i}(·|s_in) and record log-prob
- s', rewards, done, info ← Env.step(actions)
- append info["load"] to recent_loads; store per-step in ep_trace
- For each agent i: compute discounted returns G_t^i (γ), normalize across steps, compute REINFORCE loss:
L_i = −(1/T) ∑_t log π_{θ_i}(a_t^i|s_t_in) · Ĝ_t^i
and perform optimizer step on the chosen member (opts[i][chosen_idx[i]].step()).
- Append per-episode QoE/latency/cost to per_episode_metrics and optionally save ep_trace.
- Co-evolutionary update (if (ep % EVOLVE_FREQ) == 0 and ep < MARL_EPISODES):
• For each agent i: evaluate every member over EVAL_PER_MEMBER episodes (using env_factory) to get pop_scores[i].
• population[i] ← evolve_population(population[i], pop_scores[i]) where evolve uses:
- selection: tournament k = TOURNAMENT_K,
- crossover: parameter averaging of two parents,
- mutation: add Gaussian noise (std = MUT_STD).
• Re-initialize optimizers for new members.
- Online LSTM rolling update (only if MODE == Online and sufficient recent traces):
• Every p (= LSTM_UPDATE_FREQ_EPISODES) episodes, build small dataset from recent_trace_buffer
and fine-tune LSTM for a few epochs (low lr, few epochs).

1.7 Final evaluation of best policies:
- For each agent choose best member ← argmax over pop_scores (or evaluate final population) and collect EVAL_EPISODES of greedy rollouts (a = argmax π).
- Compute trained mean±std metrics (QoE, latency, cost).

1.8 Save run artifacts:
- `summary_run{r}.csv` (baseline vs trained mean±std)
- `per_episode_metrics_run{r}.csv` (QoE, latency, cost per training episode) — optional
- `lstm_comparison_rmse_run{r}.csv`
- `population_scores_run{r}.csv`
- training reward traces and plots (PNG)

2. Aggregation across runs:
-For each metric compute mean ± std across N_RUNS (produce `aggregate_results _mode_{MODE}.csv`).
- Produce final plots (convergence, bar charts with error bars) and tables (RMSE comparison).
Ablation study:
- Repeat the above for MODE ∈ {NoPred, PretrainOnly, Online} and compare:
• NoPred: agents have no LSTM signal (input_dim=5)
• PretrainOnly: LSTM pre-trained on baseline and fixed during MARL (input_dim=6)

• Online: LSTM pre-trained and periodically fine-tuned online (input_dim=6)

Remarks:

- Reporting: always present mean ± std across runs and perform statistical tests (paired t-test or bootstrap CI) where applicable.

- Reproducibility: log seeds, hyperparameters, and saved artifacts per run for traceability.

3-4) System Architecture

In this study a simulation-experimental model named Co-EvoDT is proposed, in which three independent agents or digital twins—content, infrastructure, and user—interact simultaneously and co-evolutionarily within a simulated metaverse environment. In general, each agent possesses an independent policy network trained with the REINFORCE algorithm. This co-evolutionary framework implements a dual hybrid of online parameter updates based on the policy-gradient (REINFORCE) and periodic population evolutionary steps. For each agent (content, infrastructure, user), a set of policy-network members is maintained, and in every episode, a sampled member is executed. To enable rapid and local policy correction, only that sampled member is updated online by computing the discounted return, applying variance reduction and normalization, and performing an update step with an adaptive optimizer. At specified training intervals (the co-evolutionary step), an evolutionary stage is executed that includes evaluating each member over several episodes to estimate fitness, assessing population members, tournament selection of parents, parameter aggregation of parents via parameter averaging, and applying Gaussian mutation to weight parameters; produced offspring replace weak members and the corresponding optimizers of the new members are re-initialized. During the evaluation of each member, other agents may adopt random actions so that the effect of simultaneous combinations is reduced and each member's competence can be measured to some extent independently. This mechanism is implemented in the algorithm by using stochastic policies for non-evaluated agents during the population-evaluation phase.

In the periodic co-evolutionary steps, the Adam optimizers of new members produced by the evolutionary operators (tournament, parameter aggregation, and Gaussian mutation) are re-initialized with default parameters. This approach prevents adverse effects from accumulated momentum terms and variance inherited from parent members, allowing each new member to start the learning process from an optimized starting point. Conversely, the optimizers of existing population members that were not affected by the evolutionary operators remain unchanged so as to preserve their valuable adaptive parameters.

Moreover, to cope with the high-variance gradients challenge in the REINFORCE algorithm, optional gradient clipping with a fixed threshold is used. This mechanism, by soft-limiting gradients, prevents extreme fluctuations in network-weight updates and ensures training stability across consecutive episodes.

Since evolution is restricted to the weights of the policy networks, network architectures, the sequential-predictor architecture, and high-level hyperparameters are kept fixed to preserve comparability and the reproducibility of results. This gradient–population hybridization enables the retention and exploration of behavioral multimodality, prevents mode collapse, and combines long-term (evolutionary) exploration with short-term (gradient) adaptation. Additionally, the sequential predictor's (LSTM) output is injected as an augmented feature into each agent's state vector so that policies can benefit from a future-load forecast. In online mode, this predictor is periodically fine-tuned (typically every 10 episodes) with new data at a low learning rate for a limited number of epochs. All of these measures improve decision-making toward reducing latency and cost and increase resilience in complex and non-stationary metaverse environments. Empirical results show that the combination of parametric reinforcement learning, population evolutionary mechanisms, and sequential predictive information materially reduces average latency and cost compared to classical MARL, while strengthening stability and convergence.

The objective of all three agents, simultaneously, is to optimize the quality of experience, reduce latency, and control cost; exclusively, the objective of the content and user twins/agents is to maximize

the quality of experience and minimize latency, while the infrastructure agent's goal is to balance the three factors.

3-4-1) Simulation Environment and State Dynamics Model

The symbols used in the research are as follows:

Table 1. Symbols

Symbol	Explanation
Lt	System load
vt	Virality/velocity
ct	Capacity (Infrastructure Capacity / Resource Availability)
qt	Content quality
latt	Latency
τt	Time-step/normalized time
rti	Reward for agent i at time t
w	Weight of the reward function
Lt	System load
SEED	Random seed for reproducibility
POP_SIZE	Population size of policies for each agent
EVOLVE_FREQ	Frequency of evolutionary stage execution (episode)
EVAL_PER_MEMBER	Number of evaluation episodes per member in evolution stage
TOURNAMENT_K	Group size in tournament selection
MUT_STD	Standard deviation of Gaussian fluctuation for mutation
K / EPISODE_LEN	Episode length
P / LSTM_UPDATE_FREQ_EPISODES	Online update frequency

3-4-1-1) Mathematical Form of the State

In this system, the environment is modeled as a Markov Decision Process (MDP) with discrete time steps, and the length of each episode (a full simulation run) is set to 50 time steps:

$$T=k$$

$$T = \{\text{EPISODE}_{\text{LEN}}\} = 50$$

The environment state at time step t is defined as a five-dimensional vector: $\mathbf{s}_t=[v_t, \ell_t, c_t, q_t, \tau_t]^\top$

where:

v_t : content virality
ℓ_t : system load
c_t : infrastructure capacity
q_t : intrinsic content quality
τ_t : normalized time within the episode

For initialization, the infrastructure capacity at the start of each episode is fixed ($c_0 = 0.5$) and, to introduce scenario diversity, the other initial components are sampled randomly from a uniform distribution over [0,1]:

$c_0=0.5$

$v_0 \sim U(0,1)$

$\ell_0 \sim U(0,1)$

$q_0 \sim U(0,1)$

Initial values are sampled from U(0,1); then for variable c (capacity), the initial value ($c_0 = 0.5$) and operational bounds are clearly specified in the equations:

clipped between 0.1 and 1.5.

3-4-1-2) Action Space and Action Mapping

In this model, we define three independent agents (content, infra, user). At each time step t, each agent i chooses a discrete action from the set {0,1,2}. These discrete actions are converted into continuous quantities to affect the system. The conversion for each agent is performed with specific formulae and fixed coefficients as follows:

Content agent:

promotiont=αp·atcontent, where αp=0.05

Infrastructure agent:

infra_alloct=αi·atinfra, αi=0.08

User agent:

user_engaget=αu·(atuser−1), αu=0.02

A negative user_engage indicates a reduction in engagement relative to the reference action (action = 1).

Table 2 shows the numerical effect of each agent's action (content, infra, user). Higher actions for content and infrastructure have stronger positive effects. For the user, the middle action represents normal baseline: lower action means negative engagement and higher action means higher positive engagement.

Table 2. Effects of Actions Chosen by Each Agent

Agent	Action αt	Impact Value	Simple Interpretation
Content	0	0.05×0=0.00	No advertisement
	1	0.05×1=0.05	Low advertisement
	2	0.05×2=0.10	High advertisement
Infrastructure	0	0.08×0=0.00	No capacity increase
	1	0.08×1=0.08	Slight capacity increase
	2	0.08×2=0.16	Significant capacity increase
User	0	0.02×(0−1)=−0.02	Less than normal interaction
	1	0.02×(1−1)=0.00	Normal interaction
	2	0.02×(2−1)=0.02	More than normal interaction

3-4-1-3) Dynamics Equations and Noise

This section describes how the next-step state depends on the current state and actions (i.e., how variables influence one another). Small Gaussian noises are used to simulate uncertainty. Virality, capacity, and latency are updated according to the following formulae:

- Virality (activity) (v):

vt+1 =clip(vt(1+ϵtv)+promotiont ·(1−0.2(1−ct)), 0.01, 1.0),ϵtv∼N(0,0.022)

where vt represents the current virality, ϵtv stands for a small Gaussian random noise, promotion is advertising resources, and clip limits the value between 0.01 and 1.0.

- Capacity (infrastructure capability) (c):

ct+1 =clip(0.98·ct + infra_alloct, 0.1, 1.5)

clip limits the value between 0.1 and 1.5.

infra_alloct is the newly allocated infrastructure resources at the same time step.

The factor 0.98 causes about a 2% decay in capacity each step, which is then reinforced by new allocations.

- Latency:

The next latency is a combination of current latency + effect of user-side activity + user engagement − mitigating effect of infrastructure allocation + random noise.

$\epsilon t\ell$ is a small random noise (unpredictable network fluctuations)

clip: limits result between 0.0 and 2.0

user_engaget: intensity of user engagement (e.g., number of clicks, concurrent sessions)

infra_alloct: amount of infrastructure resources allocated to reduce latency

$\mathrm{lat}_{t+1} = \mathrm{clip}\left(\mathrm{lat}_t + \beta_\ell(\mathrm{v}_{t+1} + \mathrm{user}_{\mathrm{engage}_t}) - \mathrm{infra}_{\mathrm{alloc}_t} + \epsilon t^{\ell}, 0.0, 2.0\right)$,$\epsilon t\ell \sim N(0,0.012)$

$\beta_\ell = 0.5$

3-4-1-4) Observable Outputs (Performance Metrics)

This section defines the macro-level variables used to evaluate system performance and behavior: responsiveness (latency), user experience, and system economic analysis (cost).

- Latency:

$\varepsilon \cdot \beta\ell$: a small noise for uncertainty.

$\mathrm{latency}t = \mathrm{clip}(\frac{\ell_t}{c_t} + \varepsilon \cdot \beta\ell, 0.0, 2.0), \quad \beta\ell = 0.5$

- Quality of Experience (QoE):

$\mathrm{QoEt} = \mathrm{clip}(qt - \gamma q \cdot \mathrm{latency}t + \epsilon tq, 0.0, 1.0), \quad \gamma q = 0.6, \epsilon tq \sim N(0,0.052)$

qt: base quality

γq: sensitivity coefficient

γq·latencyt : penalty of latency on QoE

ϵtq: a small random noise

- Cost:

$\mathrm{costt} = \kappa i \cdot \mathrm{infra\_alloct} + \kappa p \cdot \mathrm{promotiont}, \quad \kappa i = 0.8, \ \kappa p = 0.5$

κi and κp represent the weighting coefficients for infrastructure and promotion costs, respectively.

3-4-2) Agent Design and Learning Architecture

3-4-2-1) Policy Architecture (PolicyNet)

Each agent in the system has an independent policy neural network (MLP) that, given the state vector (environment inputs), computes the probability of selecting each action. The network architecture is:

- Input: 5 neurons (base case) corresponding to the state vector dimensions. In the state-augmented configuration with LSTM prediction, this input increases to 6 neurons.
- Hidden layer 1: 64 neurons + ReLU (models nonlinear complex features of environment states)
- Hidden layer 2: 32 neurons + ReLU (compresses information and extracts salient features)
- Output: 3 neurons to produce logits for each action and a probability distribution over actions

Population of policies and updates:

- For each agent, a set of PolicyNets is maintained.

- At the start of each episode, a random member is selected and updates are applied only to that member.
- Each member has its own Adam optimizer so it can learn and preserve its particular behavioral state.
- This design enables simultaneous population exploration and local gradient optimization.
- During co-evolutionary steps, optimizers of new members are re-initialized while optimizers of existing members are preserved.

Sampling and evaluation procedures:

- Action selection during training: during training, actions are sampled stochastically from the softmax distribution and the log-prob of each action is extracted for the REINFORCE objective (with ε = 1e-8 added to avoid log(0)).
- Action selection during evaluation: during evaluation, actions are chosen deterministically with argmax over the softmax output.
- LSTM predictor: The LSTM predictor is trained and evaluated independently (LSTM training data are sequences of system load with fixed length LSTM_SEQ_LEN).
- State-augmented mode: the LSTM's prediction $\left(\hat{L}_{t+1}\right)$ is injected as the sixth input to the policy networks. In Online mode, the LSTM is periodically fine-tuned (every p episodes) with new data.
- This parameter p is set independently from episode length k so that the decision horizon and the predictor update frequency remain decoupled.
- Implementation note: using raw logits and categorical distributions can make log-prob computation more stable; however, the current implementation is consistent with the described code and performs correctly.

3-4-2-2) Training Algorithm (REINFORCE)

Each agent learns via the policy-based REINFORCE (Monte Carlo) algorithm to maximize its cumulative reward.

- Discounted return for agent i:

$$G_i^t = \sum_{k=t}^{T-1} r^{k-t}\, r_k^i \,. \quad \gamma = 0.99$$

rki: reward of agent i at step k

γ: discount factor (0 < γ < 1), determines the importance of future rewards (γ close to 1 favors future rewards, while γ close to 0 favors immediate rewards)

- Normalization of returns:

To stabilize learning, episode returns are variance-reduced and normalized (zero mean, unit variance) before being used in the gradient:

$$\tilde{G}_t^i = -\frac{G_t^i - \mu\, G^i}{\sigma G^i + \varepsilon}$$

G_i^t: discounted return

μ G^i: mean of returns for agent i in the episode

σG^i: standard deviation of returns for agent i

ε = 1e-8 to avoid division by zero

- Loss function for agent i:

REINFORCE increases the probability of high-return actions; equivalently, the loss decreases when a high-return action is chosen, so the algorithm reinforces that action.

$$L(\theta^i) = -\frac{1}{T}\sum_{t=0}^{T-1} \log\pi_{\theta^i}\left(a_t^i \mid s_t\right) \cdot \tilde{G}_t^i$$

θi: weights/parameters of agent i's network

ati: action taken by agent i at time t

$\pi_{\theta^i}\left(a_t^i \mid s_t\right)$: probability assigned by agent i's network to action ati in state st

st: environment state

$\tilde{G}_t^i$: normalized return

- Approximate gradient of the objective:

To increase the probability of high-return actions, the approximate gradient of the objective is computed:

$$\nabla_{\theta^i} J(\theta^i) \approx \frac{1}{T}\sum_{t=0}^{T-1} \nabla_{\theta^i} \log\pi_{\theta^i}\left(a_t^i \mid s_t\right) \cdot \tilde{G}_t^i$$

∇_{θ^i} : Gradient with respect to network weights

$J(\theta^i)$: Objective function for maximization (average return)

After computing this approximate gradient for each agent, the network weights are updated using the Adam optimizer. Adam adaptively sets update steps using moment and variance estimates of the input gradients, resulting in more stable learning and faster convergence than fixed-step updates. Adam is used with default parameters (betas = (0.9, 0.999), eps = 1e−8, weight_decay = 0) and a policy learning rate (POLICY_LR = 5×10 −3). A separate Adam optimizer is used per agent, and gradient clipping is optionally applied to prevent sudden weight oscillations and to preserve training stability.

3-4-2-3) Reward Function

Penalty-type variables are given negative signs so that their effect reduces the final reward:

rti = wQoEi·QoEt + wcosti·(−costt) + wlati·(−latencyt) + wpromoi·(−promotiont)

wQoEi ،wcosti ،wlati ،wpromoi: weights for each component for agent i

Sample chosen weights and coefficients (experimentally tuned based on preliminary tests): w_QoE = 1.1, w_promo = −0.6, w_latency = −0.15, w_cost = −0.9

For learning stability, returns are normalized before use to avoid large gradient variance.

3-4-3) Sequential Predictor (LSTM)

An LSTM network is trained to predict the next-step load (t+1). Each input sample contains 8 consecutive values (seq_len = 8) and the model predicts the next-step load. Model architecture:
One LSTM layer with hidden_size = 32

- One fully connected layer mapping the LSTM output to a scalar
- The predictor operates in three modes: NoPred (no prediction used), PretrainOnly (use of a pre-trained, fixed LSTM), and Online (periodic fine-tuning with new data).

LSTM input: sequence ℓt−7 … ℓt (seq_len = 8). The point forecast ($\hat{L}t$) is computed by the LSTM after each step or episode and, in state-augmented mode, is fed as the sixth state component ($\hat{L}t$) into the policy networks.

Data split: training (70%), validation (15%), test (15%). MARL evaluation sequences are generated independently of LSTM training data to avoid data leakage. Given parameters BASELINE_TRACE_EPISODES = 160, EPISODE_LEN = 50, seq_len = 8, the number of training sequences is computed as:

sequences = BASELINE_TRACE_EPISODES × (EPISODE_LEN − seq_len) = 160 × (50 − 8) = 6720.

Loss function: the model is trained with Mean Squared Error (MSE) to minimize prediction error.

Training hyperparameters:

- Learning Rate: Aims to prevent drastic fluctuations and ensure training stability by setting a minimum value for weight changes at each training step:

LSTM_LR = 1×10−3

- Training Epochs: Determines the number of times the training data is presented to the model for convergence and complete learning.

LSTM_EPOCHS = 20

In Online mode, the LSTM is fine-tuned every p episodes (LSTM_UPDATE_FREQ) with a lower learning rate (ONLINE_LSTM_LR) for a limited number of epochs (ONLINE_LSTM_EPOCHS).

To inject realtime prediction into the policy network, the prediction ($\hat{L}t$) from the trained LSTM on the latest observed sequences can be extracted and concatenated to the state at action selection time.

3-4-4) Simulation Execution Steps

3-4-4-1) Baseline Trace Generation

To build baseline outputs, 160 episodes were executed with completely random actions and the traces (histories of states, actions, outputs) were stored. The average baseline metrics (QoE, Latency, Cost) were recorded as reference.

- LSTM pretraining: Baseline data were used to train the LSTM with sequence length 8 (LSTM_SEQ_LEN). The architecture (LSTM(hidden=32) → FC) was trained for 20 epochs (LSTM_EPOCHS) with lr = 1e-3 and MSE loss. Evaluation on a held-out set used RMSE and comparisons were made with baseline methods (Naive last-value, ARIMA, ExpSmooth).
- MARL training: The REINFORCE-based multi-agent algorithm was executed to learn operational policies. Training episodes in most experiments numbered 200. In every 40 episodes, an evolutionary/co-evolutionary step was performed (evaluate members, tournament selection, parameter aggregation, Gaussian mutation). Convergence criterion: change in average episodic reward over the last 20 episodes < 1×10−3.

Final evaluation: rained policies were tested greedily (select action with highest probability) over 80 episodes. Mean QoE, latency, and cost were extracted and compared against baseline.

Main experiment settings:

SEED = 42; MARL_EPISODES = 200; K = 50, EPISODE_LEN = 50; POLICY_LR = 5e−3; LSTM: seq_len = 8, epochs = 20, lr = 1e−3; EVOLVE_FREQ = 40

4) Findings

To measure the effect of forecasting, three scenarios were examined:

- MARL without prediction
- MARL with a pre-trained, fixed LSTM
- MARL with online LSTM (rolling update), periodically fine-tuned with new data

Using the LSTM as an online predictive signal significantly reduced Avg_Latency and Avg_Cost compared to the no-prediction case. Sensitivity analysis showed that ±30% changes in the α coefficients produced noticeable changes in QoE, Latency, and Cost.

The outputs of the proposed Co-EvoDT model (co-evolving digital twins) based on MARL and the LSTM predictor are presented. Costs in this study are relative/simulated; mapping to real-world units requires platform economic data.

4-1) Convergence Trends and Learning Curves of Agents

The presented chart shows the complex co-evolutionary interaction among content, infrastructure, user agents, and the MARL algorithm's success in coordinating them to achieve macro objectives during 200 training episodes. Small fluctuations in the learning curves can originate from co-evolutionary steps every 40 episodes (EVOLVE_FREQ) that re-evaluate the population and replace weak members.

- Content agent: from the beginning of training episodes it experiences a rapid ascending increase in reward, indicating effective learning of optimal promotion policies.
- Infrastructure agent: initially punished by negative rewards due to the cost-oriented reward matrix, but gradually improves as it learns stable, optimal allocations under dynamic environmental conditions.
- User agent: converges to a stable level with positive values, indicating continuous improvement of user engagement patterns with the system.

Overall, the gradual convergence of all three agents' curves indicates the proposed framework's efficacy in stable multi-agent learning and the metaverse's capability to provide a dynamic substrate for balancing agents' conflicting interests.

Figure 1. Training: Smoothed Episodic Return per Agent

4-2) Improvement in Performance Metrics

The following chart shows simultaneous improvements in quality, latency, and cost between the random policy and the trained policy during evaluation episodes.

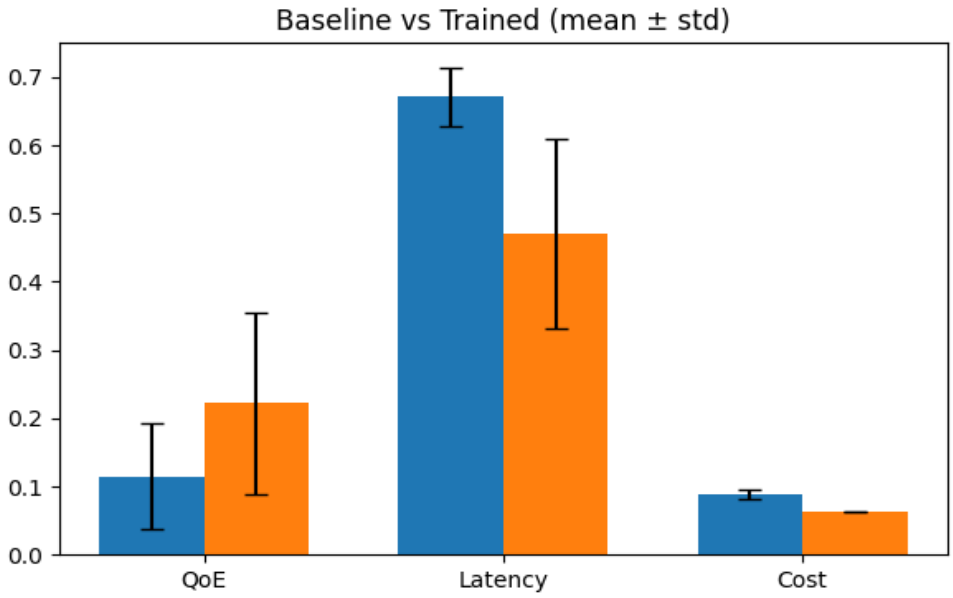

Figure 2. Average Performance: Baseline Model vs. Trained Model

Reported values in the summary table are means and accompanying standard deviations (std).

Table 3. Summary

Scenario	Avg QoE	Std QoE	Avg Latency	Std Latency	Avg Cost	Std Cost
Baseline	0.114805	0.077324	0.670913	0.043187	0.088392	**7.327893e-03**
Trained	0.221766	0.132794	0.471316	0.138777	0.064000	**1.387779e-17**

- Quality of Experience (QoE): as shown in the table, mean QoE increased from 0.114805 under the random policy to 0.221766 under the trained policy (≈ 93% improvement). This reflects the simultaneous optimization of content-production operations and resource-allocation strategies.
- Latency: average system latency decreased from 0.670913 for the random policy to 0.471316 for the trained policy (≈ 29% improvement). This notable responsiveness improvement confirms the agents' effectiveness in smarter infrastructure resource management.
- Cost: mean operational cost decreased from 0.088392 (random policy) to 0.064 (trained policy) (≈ 27% improvement). This result attests to the agents' effectiveness in using resources efficiently and improving economic performance.

The simultaneous improvement of metrics validates the model's capability in multi-objective optimization and demonstrates the potential of co-evolving digital twins beyond mere real-time mirroring toward self-improving live systems.

4-3) LSTM Predictor Performance in Load Forecasting

A figure illustrates the LSTM predictor's difference versus three baseline methods in system load forecasting (x-axis: samples, y-axis: load value). The "True (holdout)" blue line denotes real load behavior—nearly constant close to 2.0. Other colored lines are model predictions indicating several drops (~30, ~75, ~115, ~155) where load decreases approximately to 1.4–1.9.

In terms of RMSE, the LSTM significantly outperformed the three reference methods. The LSTM reduced error by about 53.2% relative to ARIMA and about 43.1% relative to the Naive method. Against

Exponential Smoothing, it achieved an approximately 88.1% reduction in RMSE. Absolute error values corroborate this trend: ARIMA = 0.0248, Naive = 0.0204, LSTM = 0.0116, Exp = 0.0971. Overall, LSTM is the most accurate model by a considerable margin.

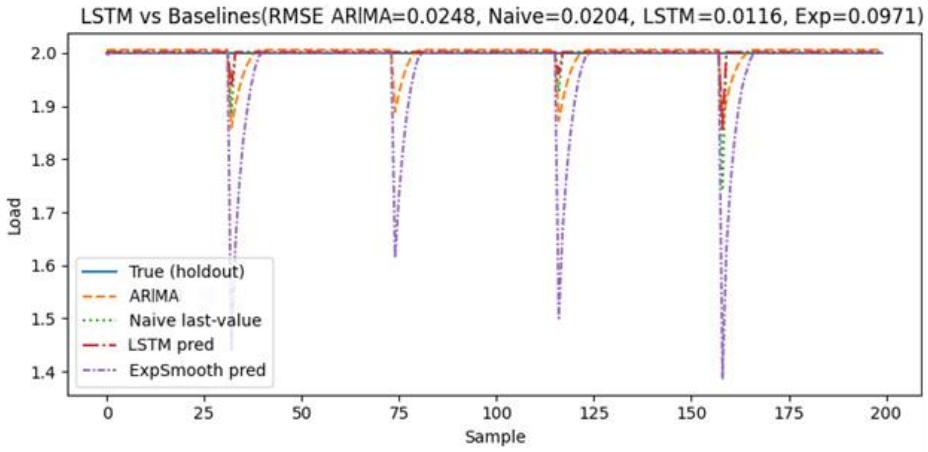

Figure 3. LSTM vs. Three Baselines in Load Forecasting

4-4) QoE Variations Over Time Under the Optimal Policy

This chart shows real-time QoE behavior during a sample episode under the trained policy. QoE initially attains higher values (≈ 0.3–0.45), then exhibits mild fluctuations and a mid-episode relative decline. These temporary drops are usually due to load peaks or increased content virality. By the episode's end, QoE stabilizes at approximately 0.14–0.25. The optimal policy that uses load forecasting (LSTM) and predictive resource allocation prevents QoE from collapsing to zero, introduces upward trends at certain points, reduces deep drops, and improves partial recoveries, resulting in overall higher and more stable QoE than the baseline. This behavior reflects the system's shift from reactive to predictive management and the effectiveness of forecasting and allocation mechanisms in preserving user experience.

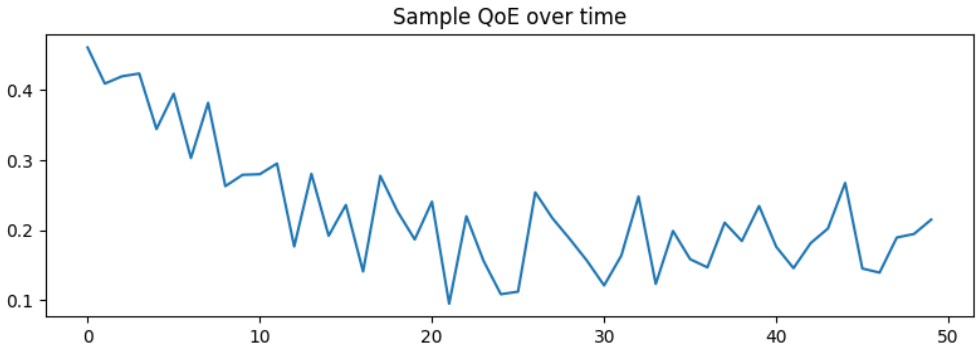

Figure 4. QoE Variations Over Time Under the Optimal Policy

4-5) System Latency Variations Over Time Under the Optimal Policy

Latency follows a gradual and relatively uniform increasing trend (from ≈ 0.17 at the start to ≈ 0.68) and reaches a saturation-like plateau at approximately time 21, remaining stable until the episode end—indicating effective traffic and resource management. Stabilization of latency near a ceiling reflects reaching capacity limits and the system's predictive behavior under load.

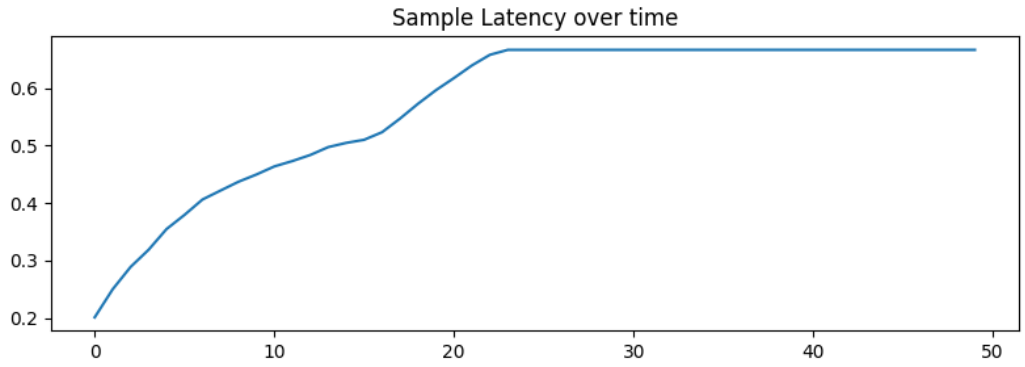

Figure 5. System Latency Over Time Under the Optimal Policy

4-6) Content Virality Variations Over Time Under the Optimal Policy

According to the plot, content virality at the start (time zero) is ≈ 0.205. Over time, it intensifies with short-term peaks concurrent with increased user requests. In time windows 10–50, it reaches successive peaks (≈ 0.230–0.235). After peaks subside and sudden drops are prevented while QoE stability is preserved, virality decreases rapidly and then may rise again. Ultimately, the optimal promotion policy controls abrupt changes and virality, preventing sudden fluctuations and maintaining QoE stability toward ≈ 0.205.

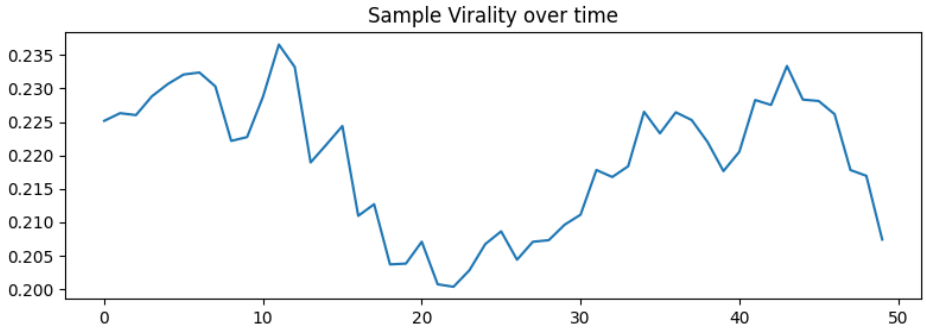

Figure 6. Content Virality Over Time Under the Optimal Policy

4-7) System Load Variations Over Time Under the Optimal Policy

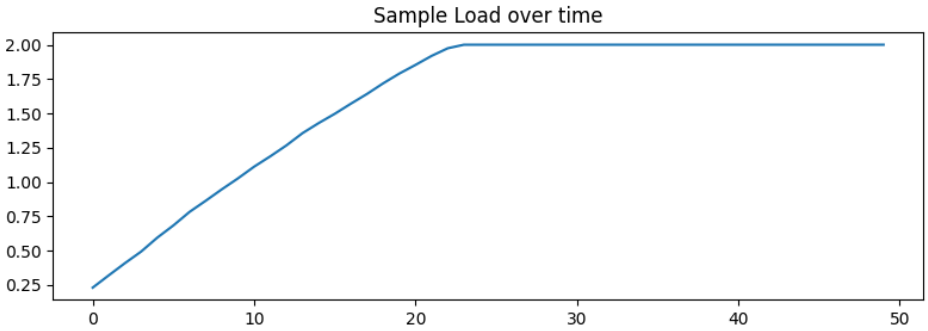

Figure 7. System Load Over Time Under the Optimal Policy

In the plot, system load grows roughly linearly from ≈ 0.20 at the beginning to ≈ 2.0 (operational ceiling) by around time 21. After reaching this level, load remains stable. The predictive allocation and load-distribution policy enable controlled growth, the prevention of bottlenecks and critical drops, and the stabilization of system performance near a sustainable capacity boundary.

Summary

Increased latency directly causes QoE degradation; however, the coordination of infrastructure resource allocation with sequential forecasting (LSTM) and joint MARL policies substantially mitigates latency's negative impact on QoE. Although higher virality and content promotion are directly associated with increased system load and cost, intelligent predictive resource allocation enables load control at manageable levels and prevents sharp latency increases during peak demand. The apparent conflict among objectives (reducing cost and latency vs. maximizing QoE) can be moderated via the co-evolution of digital twins and multi-objective policies across the three agents.

5) Conclusion and Recommendations

Despite theoretical advances in the co-evolution of digital twins (Bersani et al., 2022; Fan et al., 2025; Feng et al., 2025; Tang, 2024), to date, no approach has reported—in real-time metaverse scenario simulations—an operational combination of REINFORCE-based multi-agent reinforcement learning, LSTM sequential forecasting, and gradient–population co-evolutionary update mechanisms for the simultaneous optimization of QoE, latency, and cost in this ecosystem.

In contrast, this paper was able to simultaneously improve three key indicators of media digital transformation within a simulated metaverse environment. The effectiveness of this framework is validated not only by a 93% improvement in user quality of experience, a 29% reduction in latency, and a 27% reduction in cost, but this achievement is particularly important in the multi-objective optimization problem in the dynamic and highly uncertain metaverse environment, where coordination among different agents and forecasting variable loads becomes problematic.

The synergy of policy coordination and LSTM predictive control improved load-forecast accuracy compared to a simple naive method by approximately 43%, relative to ARIMA by about 53.2%, and relative to Exponential Smoothing by about 88%. This predictive signal enabled proactive resource management and the mitigation of sudden latency spikes by decision-making agents. In other words, the addition of the LSTM forecast signal—and especially the online rolling fine-tuning mode—turned MARL decision-makers from reactive to predictive, and this shift played a central role in meaningfully reducing latency and cost peaks.

Thus, applying this co-evolutionary mechanism and the intelligent interaction among the three factors of content, relying on multi-faceted reward shaping and load-forecast signals, can serve as an efficient tool to reduce the inherent conflict between content promotion (increasing virality) and infrastructure management (controlling cost and latency).

According to the findings, intelligent interaction among all three factors emerged through multi-objective optimization: although increased content promotion was directly associated with higher system load and cost, infrastructure allocation guided by LSTM load forecasting and the adjustment of multi-agent reinforcement learning policies—together with the integration of sequential forecasting (to prevent peaks) and collaborative learning (for intelligent resource distribution)—reduced the conflict between latency and cost and yielded a more sustained improvement in user quality of experience. Consequently, predictive control and policy coordination among digital twins were presented as the concrete reasons for the observed synergy among the three factors.

Overall, the practical and theoretical contributions of this research can be explained across several axes: In this idea, two updating mechanisms were used simultaneously to train agents: gradient-based updates (the REINFORCE algorithm) and population evolution. The use of gradient updates based on Adam for each population member enabled the rapid, local adaptation of each policy to short-term environmental changes. In contrast, periodic evolutionary operators (such as tournament selection, parameter aggregation, and Gaussian mutation)—in addition to introducing behavioral diversity and new members into the population—allowed long-term search in the policy space and escape from local optima. This dual hybridization enabled the system to simultaneously benefit from the rapid adaptability characteristic of gradient methods and the deep, sustained exploration characteristic of evolutionary methods, thereby producing more stable convergence and greater resilience in the complex metaverse environment.

This model presents an integrated and hybrid framework that combines the control dimension via MARL and the forecasting dimension via LSTM in the form of co-evolving digital twins in the metaverse, providing an intelligent algorithmic tool for modeling timed interactions among content, infrastructure, and users. Therefore, the integration of multi-agent reinforcement learning and sequential predictors has the potential to generate an intelligent, dynamic transformation in the design of digital twins in the metaverse, provided that results are validated on real-world data volumes.

Trend analysis of reward convergence also showed that the content and user-quality agents experienced positive convergence after initial oscillations, whereas the infrastructure agent—with cost-weighted entries in the reward matrix—initially exhibited negative returns but gradually improved with the application of optimal allocation policies. This convergence behavior indicated that, in co-evolutionary frameworks, reward weighting, conflicting rewards, and scaling of penalties are decisive instruments for steering stable learning in multi-agent systems; meaning that overall convergence, emerging from interactions among independent agents (with conflicting objectives in a shared environment) and dependent on the balance of rewards and the interplay between temporal prediction and policy-driven control, constitutes a step toward the development of coordination theories in multi-agent systems.

From a managerial and policy perspective, the findings of this research can form the basis for designing intelligent predictive resource-management systems in media and metaverse platforms so that, through the deployment of co-evolving digital twins and the application of optimal policies, user quality of experience can be improved without increasing costs. Moreover, the use of co-evolving digital twins can serve as a strategic tool for technology policymakers to design scalable and efficient infrastructures.

Although the proposed framework can serve as a basis for multi-agent optimization in the metaverse, its simulation comes with limitations, including: lack of access to real metaverse environments and data (which are evolving) and reliance on a simplified simulation environment; constraints on the number of tested agents and the scalability of experiments due to hardware limits of the simulator host; limited experimental scale (the number of agents and length of training episodes); limitations in LSTM hyperparameter tuning; lack of explicit uncertainty quantification; and conversion of costs to real-world units. Accordingly, more realistic scenario modeling by expanding the model to more agents, comparing different MARL algorithms (such as MAPPO, PPO), generalizing the model to domains such as smart cities and interactive media, evaluating sensitivity to reward-matrix design and weightings, are suggested future research paths that arise from these limitations.

Although the Co-EvoDT idea achieved commendable simultaneous optimization of QoE, latency, and cost in the simulated environment, transferring this model to a real metaverse platform will involve multiple challenges. Environmental non-stationarity can be considered one of the primary challenges, because in a real metaverse, user behavior, content consumption patterns, network traffic frequency, and interactive demand types dynamically and unpredictably fluctuate—all in multidimensional data volumes. This leads to shifts in the input data distributions of LSTM models and to changes in the dynamic environment from the viewpoint of MARL agents. In such conditions, pre-trained models may rapidly become obsolete. Therefore, to address this challenge, solutions such as broad pre-training across a wide range of states using domain randomization, combined with continual learning mechanisms in the LSTM architecture, are proposed so that rapid fine-tuning can be performed in response to distributional shifts.

Delay in receiving real feedback can be considered another challenge. Given the urgent need for immediate feedback access in conducting other stages of simulation, this issue in the metaverse may face significant delays. Because of the expansiveness of the metaverse ecosystem, collecting sensor data from users, various connected objects, dispersed infrastructure components, processing data, and updating digital twins require massive imaging and video processing workflows. This latency can be particularly problematic for online-update-based algorithms, such as the LSTM Online mode and REINFORCE gradient updates, because policies are updated based on information that no longer represents the current system state. To mitigate this effect, hybrid predictive architectures that can forecast multiple possible future states of the environment in short horizons can be used, even in the

absence of fully up-to-date data. Additionally, the partial modeling of environmental dynamics to generate estimated reward signals under delay, time-alignment mechanisms, and precise timestamping between network telemetry and user telemetry are essential.

Another challenge is model scalability in the real world. As this model is designed around three factors and the complete metaverse ecosystem will contain thousands of independent agents (such as users, edge servers, content providers, and various services), the exponential increase in the dimensionality of state and action spaces, as well as the complexity of inter-agent interactions, will impractically increase the computation and resources required for training and inference. To overcome this limitation, hierarchical architectures are proposed in which agents can operate at different levels of abstraction (micro and macro). On the other hand, using transfer learning methods to transfer knowledge from a simplified simulation environment to a complex real environment may provide a viable path forward.

Given the proposed technical solutions to address these challenges, the Co-EvoDT framework emerges as a practical and effective approach for achieving intelligent, co-evolutionary management of media digital transformation in the metaverse of the future.

References

AlBalkhy, W., Karmaoui, D., Ducoulombier, L., Lafhaj, Z., & Linner, T. (2024). Digital twins in the built environment: Definition, applications, and challenges. *Automation in Construction, 162*, 105368. https://doi.org/10.1016/j.autcon.2024.105368

Alvi, M., Dutta, H., Minerva, R., Crespi, N., Raza, S. M., & Herath, M. (2025). Global perspectives on digital twin smart cities: Innovations, challenges, and pathways to a sustainable urban future. *Sustainable Cities and Society, 106356.* https://doi.org/10.1016/j.scs.2025.106356

Benaben, F., Congès, A., & Fertier, A. (2025). A prospective vision of the evolution of immersive technologies: Towards a definition of metaverse. *Technovation, 140*, 103154. https://doi.org/10.1016/j.technovation.2024.103154

Bersani, M. M., Braghin, C., Cortellessa, V., Gargantini, A., Grassi, V., Presti, F. L., Mirandola, R., Pierantonio, A., Riccobene, E., & Scandurra, P. (2022, March). Towards trust-preserving continuous co-evolution of digital twins. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)* (pp. 96-99). IEEE. https://doi.org/10.1109/ICSA-C54293.2022.00024

Chow, Y. W., Susilo, W., Li, Y., Li, N., & Nguyen, C. (2022). Visualization and cybersecurity in the metaverse: A survey. *Journal of Imaging, 9*(1), 11. https://doi.org/10.3390/jimaging9010011

Cong, A., Jin, Y., Lu, Z., Gao, Q., Ge, X., Li, Z., Rongzhou, L., Xinying, H., & Hou, L. (2025). Transfer learning-based physics-informed DeepONets for the adaptive evolution of digital twin models for dynamic systems. *Nonlinear Dynamics*, 1-28. https://doi.org/10.1007/s11071-025-11158-4

Fan, S., Tong, H., & Wang, S. (2025). A system dynamics-based hybrid digital twin model for driving green manufacturing. *Systems, 13*(8), 651. https://doi.org/10.3390/systems13080651

Feng, B., Wang, Z., Yuan, L., Zhou, Q., Chen, Y., & Bi, Y. (2025). Towards safe motion planning for industrial human-robot interaction: A co-evolution approach based on human digital twin and mixed reality. *Robotics and Computer-Integrated Manufacturing, 95*, 103012. https://doi.org/10.1016/j.rcim.2025.103012

Hady, M. A., Hu, S., Pratama, M., Cao, Z., & Kowalczyk, R. (2025). Multi-agent reinforcement learning for resources allocation optimization: A survey. *Artificial Intelligence Review, 58*(11), 354. https://doi.org/10.1007/s10462-025-11340-5

Haimes, Y. Y. (2018). Risk modeling of interdependent complex systems of systems: Theory and practice. *Risk analysis, 38*(1), 84-98. https://doi.org/10.1111/risa.12804

Li, Z., Ji, Q., Ling, X., & Liu, Q. (2025). A comprehensive review of multi-agent reinforcement learning in video games. *IEEE Transactions on Games*. https://doi.org/10.1109/TG.2025.3588809

Mousavi, S. T. , Faezy Razi, F., & Danaei, A. (2025). Medavers: A digital transformation model for media in the metaverse. *Interdisciplinary Journal of Management Studies, 19*(1), 221-246. https://doi.org/10.22059/ijms.2025.395001.677621

Novikov, R.Yu., & Zohrabyan, E.P. (2023). Digital transformation of media: challenges and opportunities. *Journal of Digital Economy Research, 1*(4), 102–125. https://doi.org/10.24833/14511791-2023-4-102-125

Olcay, K., Tunca, S. G., & Özgür, M. A. (2024). Forecasting and performance analysis of energy production in solar power plants using long short-term memory (LSTM) and random forest models. *IEEE Access*. PP(99), 1-1. https://doi.org/10.1109/ACCESS.2024.3432574

Rammel, C., Stagl, S., & Wilfing, H. (2007). Managing complex adaptive systems—A co-evolutionary perspective on natural resource management. *Ecological economics, 63*(1), 9-21. https://doi.org/10.1016/j.ecolecon.2006.12.014

Rezaeenour, J., & Karimian, R. (2024). Identifying metaverse developments in digital libraries based on library theory. *Knowledge Retrieval and Semantic Systems, 11*(39), 67-108. https://doi.org/10.22054/jks.2023.76141.1617

Rezaeenour, J., & Karimian, R. (2025). Identification of digital library indicators in metaverse environment. *The International Journal of Metaverse & Virtual Transformation (IJMVT), 1*(2), 74-94. https://mvt.artahub.ir/article_227497_ed969c594c314ad19dc784f65f9cf800.pdf

Salamattalab, M. M., Zonoozi, M. H., & Molavi-Arabshahi, M. (2024). Innovative approach for predicting biogas production from large-scale anaerobic digester using long-short term memory (LSTM) coupled with genetic algorithm (GA). *Waste Management, 175*, 30-41. https://doi.org/10.1016/j.wasman.2023.12.046

Tong, X., Bao, J., & Liu, T. (2024, August). Co-Evolution DTs: Achieving value-added cognitive digital twins across the entire lifecycle. In *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)* (pp. 3140-3146). IEEE. https://doi.org/10.1109/CASE59546.2024.10711333

Tong, X., Bao, J., & Tao, F. (2024). Co-evolutionary digital twins: A multidimensional dynamic approach to digital engineering. *Advanced Engineering Informatics, 61*, 102554. https://doi.org/10.1016/j.aei.2024.102554