DCoPilot: Generative AI-Empowered Policy Adaptation for Dynamic Data Center Operations

Minghao Li, Ruihang Wang*, Rui Tan, Yonggang Wen

Nanyang Technological University, Singapore

E-Mail: {minghao002, ruihang001}@e.ntu.edu.sg, {tanrui, ygwen}@ntu.edu.sg

Abstract—Modern data centers (DCs) hosting artificial intelligence (AI)-dedicated devices operate at high power densities with rapidly varying workloads, making minute-level adaptation essential for safe and energy-efficient operation. However, manually designing piecewise deep reinforcement learning (DRL) agents cannot keep pace with frequent dynamics shifts and service-level agreement (SLA) changes in an evolving DC. This *specification-to-policy* latency causes a lack of timely, effective control policies, which may lead to service outages. To bridge the gap, we present DCoPilot, a hybrid framework for generative control policies in dynamic DC operation. DCoPilot synergizes two distinct generative paradigms, i.e., a large language model (LLM) that performs symbolic generation of structured reward forms, and a hypernetwork that conducts parametric generation of policy weights. DCoPilot operates through three coordinated phases: (i) simulation scale-up, which stress-tests reward candidates across diverse simulation-ready (SimReady) scenes; (ii) meta policy distillation, where a hypernetwork is trained to output policy weights conditioned on SLA and scene embeddings; and (iii) online adaptation, enabling zero-shot policy generation for updated specifications via interpolation within the trained specification envelope. Evaluated across five control task families spanning diverse DC components, DCoPilot achieves near-zero constraint violations and outperforms all baselines across specification variations. Ablation studies validate the effectiveness of LLM-based unified reward generation in enabling stable hypernetwork convergence.

Index Terms—Data Center, Large Language Model, Control Optimization, Meta Reinforcement Learning

I. INTRODUCTION

The proliferation of artificial intelligence (AI) applications is spurring global computing needs, where capital investment in data centers (DCs) will expand at 21% annually through 2029 [1]. This growing investment incentivizes the development of AI-dedicated data centers (AIDCs). These facilities are characterized by high-density racks with high-performance graphics processing units (GPUs) [2]. Unlike standard computational tasks, hosting AI tasks with GPU servers is projected to raise average rack power density from $8\,\mathrm{kW}$ to $30\,\mathrm{kW}$ by 2027 [3]. At such densities, iteration-scale GPU workload behavior produces large idle-to-peak power swings that translate into tremendous thermal fluctuations [4].

AIDC's high power density and bursty compute loads increase energy consumption and reduce cooling response time for reliable operations. In practice, operators are required to manage these dynamics on minute-scale control horizons [5].
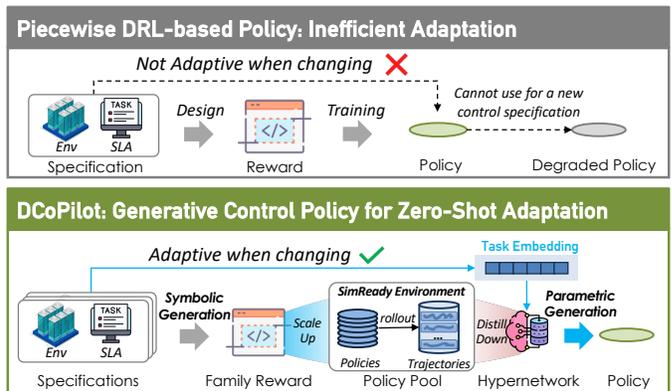

Fig. 1. DCoPilot replaces manual, piecewise DRL-based policies with a generative pipeline: an LLM formalizes operational *specifications* into a symbolic family reward, and a hypernetwork generates parametric *policy* network weights under evolving environments.

Meanwhile, frequent server installations and changing tenant service-level agreements (SLAs) within a shared data hall further affect the steadiness of coordination between information technology equipment (ITE) and cooling facilities [6]. However, manually engineering setpoints and rule-based playbooks is time-consuming and cannot keep pace with the tight response window, as it requires continual retuning across thermal–electrical variables [7]. As a cautionary example, a planned cooling system upgrade at an Equinix Singapore DC in October 2023 manually misconfigured the chilled-water system, causing overheating and bank service disruption [8]. Consequently, AIDC operations require policies that can adapt quickly and safely to evolving DC scenarios and varied tenant-specific SLAs.

Existing control policies for cyber-physical systems (CPS) can be categorized into non-generative approaches, which are designed for static environments, and generative policies that adapt to varying specifications. Among non-generative methods, expert-engineered policies rely on human expertise and experience to manually tune parameters for a specific control scenario. In DC operations, a proportional-integral-derivative (PID) controller is employed to adjust the server's internal fan speed in response to its temperature, where engineers need to manually tune the parameters to achieve a stable and efficient response [9]. Data-driven controllers under fixed objectives learn policies against pre-specified costs or constraints, but still depend on domain-specific modeling and reward shaping. Model-based approaches identify a dy-

namics surrogate and solve a constrained optimization via model predictive control (MPC) to trade off energy against temperature bounds [7], [10]. To reduce the modeling efforts, model-free approaches directly optimize expected return by interacting with the environment to train deep reinforcement learning (DRL) agents [11]–[13]. Recent work fuses physics priors with cross-environment adaptation, where an offline thermodynamics-aware model pairs with safe online fine-tuning for DRL agents [14]. Despite these advances, manual modeling and reward shaping remain expert-intensive, creating specification-to-policy latency that delays control policy deployment and may precipitate service disruptions.

In contrast, generative policies aim to overcome these limitations by synthesizing control strategies conditioned on goals, constraints, and contextual inputs. Early studies adopt the large language models (LLMs) as direct policy generators, relying on in-context expert knowledge for industrial control [15]. However, LLM hallucinations may recommend risky actions, leading to computing service outages. To ensure the safety, some LLM-based control policy generation approaches focus on industrial control code generation, involving retrieval augmented generation (RAG) with knowledge base to generate control codes based on user prompts [16], [17]. However, LLM-generated PID control is limited to setpoint tracking and cannot optimize the objectives of DC operations. Recent studies focus on using LLMs for DRL agent training. These approaches mainly focus on robotics problems using evolutionary gradient-free approaches to design reward functions [18], [19]. Unlike static embodied robots, DCs evolve due to ITE upgrades and SLA changes by tenants. The fixed parameters of LLM-guided DRL policies cannot generalize to skewed distributions in dynamic DC operations. In sum, existing policies are tailored to fixed operational specifications without considering the rapidly changing DC environments and tenant SLAs.

To bridge the gap, we introduce DCoPilot, a hybrid framework for generative control policies in dynamic DC operation. As shown in Figure 1, the core contribution of our work lies in the synergistic integration of two forms of generative AI (i.e., LLM and hypernetwork [20]) to enable zero-shot policy generation through interpolation over offline-trained policies under evolving DC specifications, including shifting system configurations and SLA setpoints. The LLM defines what to do, and the hypernetwork instantly creates a policy for achieving it. In particular, an LLM generates a shared reward form for a family of Markov decision processes (MDPs), compiling operator prompts, domain-specific metrics, and SLA bands. A hypernetwork translates the task embedding of an operational specification into deployment-ready policy weights, addressing the LLM's limitations with continuous numerical parameters.

A key design principle of DCoPilot is to shift all training overhead offline through LLM-based automation, while the online phase leverages the hypernetwork for instant policy generation, addressing the challenge of specification-to-policy latency. At its core, DCoPilot generates DC control policy in three steps: (i) simulation scale-up, (ii) meta policy distillation,

and (iii) online adaptation. During the offline simulation scale-up, DCoPilot leverages a Reward LLM to generate reward candidates and reflect on trajectories of generated candidates training over the min/max boundary operational specifications (i.e., DC system and tenant SLA) in the synthetic simulation-ready (SimReady) [21] environment. Then, DCoPilot selects the best reward form candidates, trains multiple DRL agents for varied operational specifications, and rollouts a near-optimal trajectory pool. In the meta policy distillation stage, a hypernetwork is trained on the trajectory pool, encoding each specification as a unique task embedding to capture shared policy structures. During online adaptation, new operational specifications are used as input to the pre-trained hypernetwork to generate policy weights for real-time control adjustment.

We evaluate DCoPilot against three LLM-based control baselines and one ablation setting across five task families spanning diverse DC components and operational objectives. DCoPilot achieves the lowest violation cost below $0.2\,°C$ across all task families, significantly outperforming baselines that range from $0.77\,°C$ to $8.78\,°C$, and maintains consistent temperature control with mean errors below $0.5\,°C$ across different SLA bounds and server configurations. In a 40-day operational test with two specification changes, DCoPilot keeps violations near zero throughout, while task-specific DRL methods experience large violation spikes up to $4\,°C$ during their multi-day retraining periods. Compared with meta-learning approaches on zero-shot adaptation, DCoPilot exhibits more stable control across power and temperature metrics during specification transitions, without needing any online fine-tuning. Ablation studies further show that DCoPilot's LLM-based generation of unified family reward form makes hypernetwork training stable.

Our contributions are summarized as follows:

- We propose DCoPilot, the first framework integrating symbolic reward and parametric policy network generation for dynamic DC operations, establishing a new paradigm that combines LLMs with hypernetworks for adaptive CPSs.
- We design a dual-stage generative pipeline that synergizes LLM-based reward formulation with hypernetwork-based policy generation, confining training to offline phases while enabling zero-shot online adaptation.
- Through extensive experiments across five control task families, we demonstrate superior efficiency and policy quality over baselines, with an ablation study validating a significant reduction in specification-to-policy latency.

Paper organization: §II reviews related work. §III presents preliminaries and problem definition. §IV presents the design of DCoPilot. §V presents evaluation results. §VI discusses several related issues. §VII concludes this paper.

II. RELATED WORK

A. DC Operation Optimization

DC operation optimization aims to enhance energy efficiency, reduce costs, and improve sustainability in large-scale
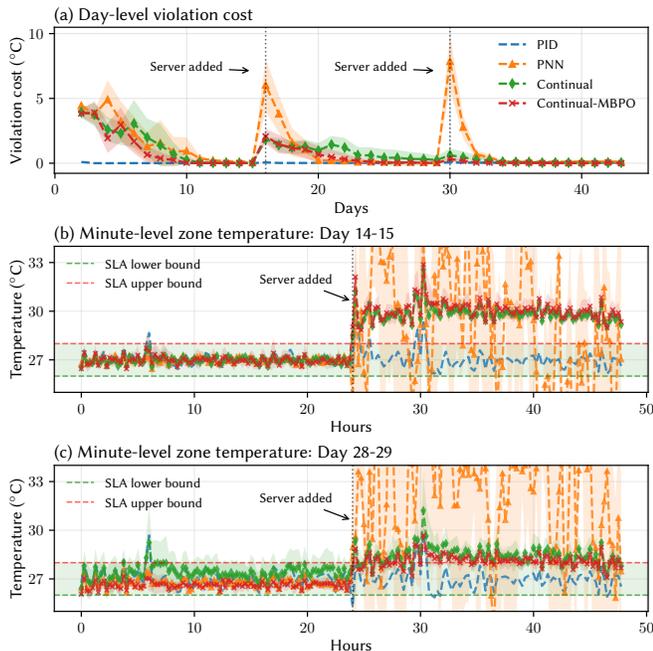
Fig. 2. Adaptation performance of continuous learning approaches at day-level and minute-level evaluations. Day-level violation cost is the time-average magnitude by which the zone temperature exceeds the SLA band, i.e., $\bar{v}_{\text{day}} = \frac{1}{N_{\text{day}}} \sum_{t \in \text{day}} \max\{0, T_t - T_{\text{U}}, T_{\text{L}} - T_t\}$, where T_{U} and T_{L} are the upper and lower SLA bounds, and N_{day} is the number of steps.

computing facilities. Traditional approaches employ model-based techniques such as MPC for regulating cooling systems [7] and evolutionary algorithms for geographical workload optimization [22]. More recent approaches apply DRL to handle dynamic DC conditions. DeepEE [23] proposes a deep Q-network framework that jointly optimizes job scheduling and cooling control in a DC simulation. Wang et al. demonstrate that DRL-based joint IT-facility optimization achieves energy reductions [24]. Sarkar et al. [25] propose a multi-agent RL digital twin environment for DCs enabling joint optimization of IT servers, cooling, load-shifting and battery storage to reduce carbon footprint. More recently, Zhan et al. employ offline RL with graph neural networks (GNN) to achieve 14-21% energy reductions by modeling physical dependencies [26]. However, these methods often require extensive domain expertise, suffer from high training costs, and lack robustness in rapidly evolving scenarios with varying SLAs and installed servers.

B. Few-Shot Adaptation for CPS Control

Adaptive CPS control includes meta-learning for few-shot adaptation, continual learning for sequential task acquisition, and model-based reinforcement learning for efficient policy learning. Model-agnostic meta-learning (MAML) learns initialization parameters enabling rapid adaptation through gradient steps, while probabilistic embeddings for actor-critic learning (PEARL) infers task embeddings from trajectories to condition policies [27], [28]. Progressive neural networks (PNN) address continual learning by adding network columns

for each task with lateral connections to prevent catastrophic forgetting [29], while model-based policy optimization (MBPO) achieves sample efficiency by learning dynamics models to generate synthetic training data [30]. These methods have shown promise across CPS-related domains, where Meta-Controller achieves few-shot imitation learning [31], LEGION enables skill recombination for lifelong learning [32], and Phyllis incorporates physical priors for efficient adaptation in DCs [14]. Figure 2 demonstrates these cases of using continual learning approaches for DC operation. In DC operations governed by minute-level decision windows, the online exploration and data collection required by few-shot adaptation may degrade computational performance or even cause outages. In sum, few-shot adaptation approaches only take fixed objective functions and cannot adapt instantaneously to burst workloads.

C. Generative AI for Adaptive Control

Generative AI encompasses diverse architectures for policy generation, including diffusion models for action sampling, hypernetworks for parametric policy adaptation, and LLMs for symbolic reasoning. Diffusion models enable multi-modal action generation [33], [34], but DC control requires constraint-satisfying behavior under SLA and actuator limits rather than diverse action sampling. Hypernetworks generate weights for target networks conditioned on task embeddings, enabling rapid adaptation across task distributions [20]. Recent work extends hypernetworks to continuous control, where conditioning on goal embeddings enables zero-shot generalization to unseen objectives [35]. However, existing hypernetwork approaches lack symbolic reasoning to interpret evolving operational requirements, making it difficult to align control policies with operational intents. LLMs have demonstrated capabilities in code generation and logical reasoning, making them suitable for generating control policies. Early studies leverage LLMs as direct policy generators through in-context learning for industrial control [15]. However, LLM hallucinations pose severe operational risks in safety-critical systems. To reduce hallucinations, recent works use knowledge bases to guide the generation of industrial control code [16], [17]. However, these studies focus on a fixed optimization objective. Another direction employs LLMs to design reward functions for DRL training. Methods such as Eureka [19] and its variants [36] employ evolutionary, gradient-free approaches to optimize reward codes for robotics tasks. However, these methods require extensive training for each new specification, creating specification-to-policy latency that is far beyond the minute-scale DC control horizons. In sum, existing generative methods tackle only parts of the control problem and cannot jointly interpret operational intent and synthesize adaptive policies.

III. PRELIMINARIES & PROBLEM DEFINITION

In this section, we introduce the preliminaries of air-cooled DC operation and formally define the problem of generative policy synthesis for dynamic DC control.
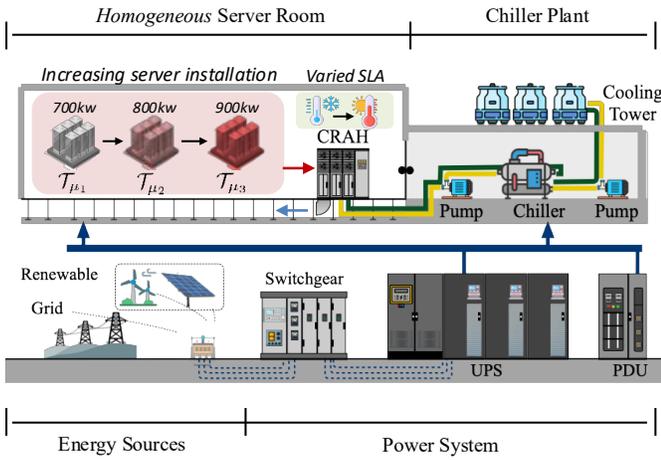
Fig. 3. General topology of DC infrastructure, including server rooms, chiller plant, electrical system, and power grid. DC operators continuously install new servers and adjust SLAs, making *homogeneous* changes over time.
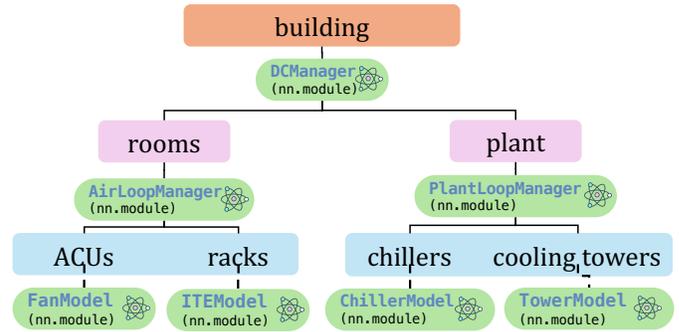

Fig. 4. The hierarchical structure of a DC SimReady scene. A DC SimReady scene consists of three-level SimReady assets. Each SimReady asset is a surrogate model trained from operational data.

A. DC Architecture & Operation

1) DC system architecture: We consider a typical DC equipped with an air-cooled system, the most widely deployed architecture. The infrastructure comprises server rooms with IT equipment (ITE), computer room air conditioning (CRAC) units, chillers, cooling towers, and the electrical distribution system. The cooling process consists of two stages. The first stage adopts the CRACs to supply cold air to the IT devices and cool the return hot air. The second stage transfers the water-carried heat by the chiller and dissipates it to the ambient. This two-stage architecture enables independent control of indoor air conditions and outdoor heat rejection. The control system manages continuous SLA setpoints to maintain ITE performance while achieving sustainability goals.

DC operators continuously install new servers to meet growing computational demands, as illustrated in Figure 3. This server expansion represents a homogeneous change, where a transformation preserves the system structure while scaling its parameterized heat gains. Specifically, adding servers to existing racks maintains the same cooling infrastructure topology, i.e., identical numbers of CRACs, chillers, and cooling towers, preserves the same state space dimensions, and keeps the same action space dimensions. For instance, upgrading from 10 to 15 servers per rack increases total heat load from $700\,\mathrm{kW}$ to $900\,\mathrm{kW}$, requiring a new optimized control policy, yet the action space and underlying physical relationships remain structurally identical.

2) Hierarchical SimReady scene: We adopt a hierarchical system model for a DC that is purely data-driven, developed in our prior work [37]. The model is a SimReady digital scene [21] that predicts future states from current states and actions. As illustrated in Figure 4, a scene description file specifies the configuration of all assets and their interconnections, resulting in a multi-level structure. The top level represents the overall `Building`. The second level separates `Rooms` from `Plant`. For the third level, we focus on `Racks` and `ACUs` within `Rooms`, and `Chillers` and `Towers` within `Plant`.

An air-cooling unit (ACU) conditions and delivers supply air for ITE. A chiller removes heat from a secondary water loop to provide chilled water to ACUs. A cooling tower rejects condenser heat via evaporative exchange.

Each component in a DC can be modeled as an asset paired with a learned surrogate trained from operational data. Coupling among assets is handled by its corresponding manager classes, where `AirLoopManager` exchanges air-side variables for server rooms, `PlantLoopManager` coordinates water-side flows for the chiller plant, and the top-level `DCManager` supervises the global simulation of the entire DC. With these SimReady assets composed through this hierarchy, the resulting SimReady scene provides a foundation for simulation and serves as a basis for generating synthetic scenes. These learned model-free surrogates preclude direct application of MPC-style controllers.

3) DRL-based cooling control: The facility controls perform with a continuous action space. For instance, the DC cooling control aims to improve power efficiency while maintaining operating temperature at a target setpoint. This operation periodically adjusts the air temperatures and mass flow rates supplied from the CRAC units, which can be modeled as a constrained MDP and solved with DRL techniques [23], [38], [39]. Formally, the DC cooling control problem is represented as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ denotes the state space capturing thermal and facility conditions, $\mathcal{A}$ is the action space consisting of continuous control inputs for active cooling facilities, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathrm{Dist}(\mathcal{S})$ models the stochastic transition dynamics, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defines the reward function that penalizes SLA violations and incentivizes sustainability goals.

B. Problem Formulation

1) Specification-driven MDP family: While DC operations involve diverse variations, we focus on operational changes that can be naturally parameterized. These variations change the MDP with every new operational specification. Installing servers modifies thermal dynamics ($\mathcal{T} \rightarrow \mathcal{T}_\mu$), parameterized by μ capturing power density. SLA updates modify reward functions ($\mathcal{R} \rightarrow \mathcal{R}_\psi$), parameterized by ψ encoding SLA
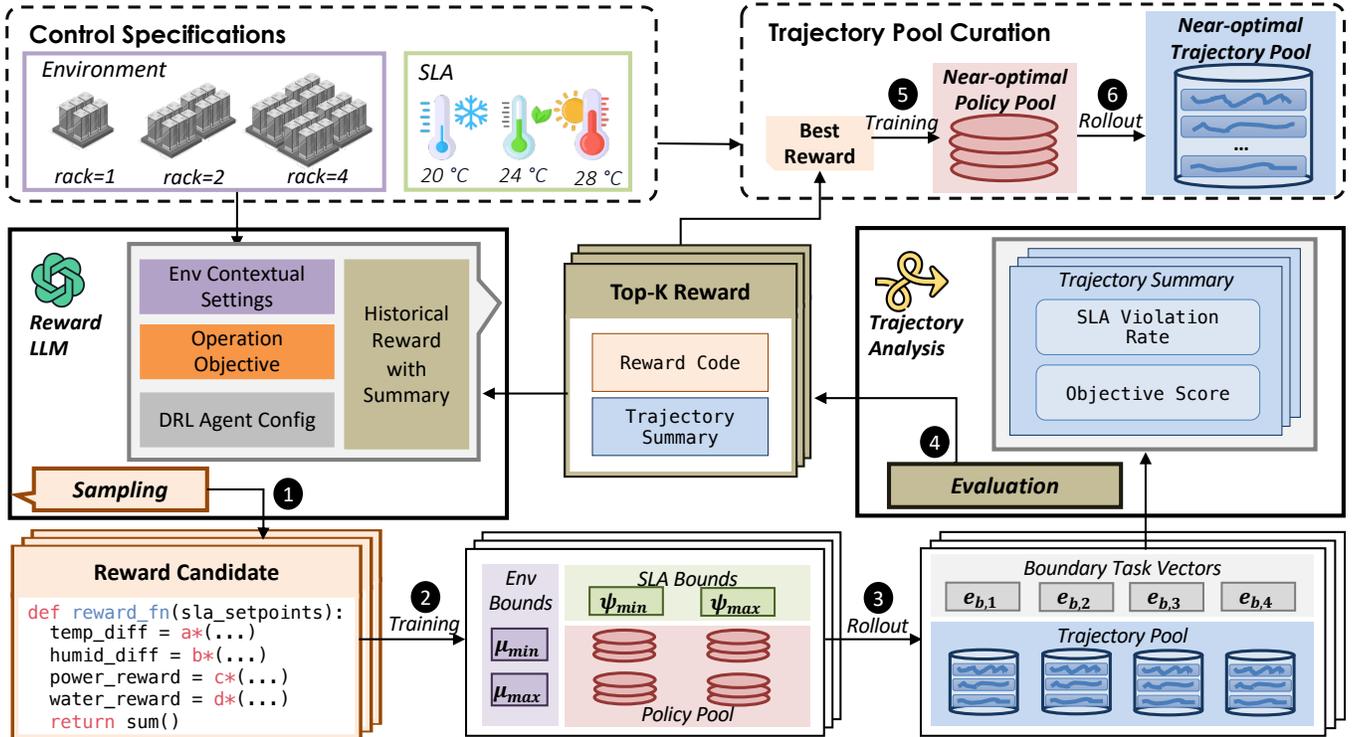
Fig. 5. Given scene G and operator objective I_{ops}, the Reward LLM generates reward code candidates (❶). Each candidate is evaluated on boundary conditions by training policies and collecting trajectories (❷-❸). Then, it ranks candidates based on SLA violation costs and objective scores, selecting top-k for evolutionary refinement (❹). The best reward form $\bar{R}^*$ trains DRL agents across sampled specifications $\mathbf{e} = \boldsymbol{\mu} \otimes \boldsymbol{\psi}$ (❺), generating near-optimal trajectory pool $\mathbf{T}$ for hypernetwork distillation (❻).

setpoints (e.g., temperature and humidity). This defines a family of parameterized MDPs:

$$\mathscr{M} = \{\mathcal{M}_i | \mathcal{M}_i = (\mathcal{S}, \mathcal{A}, \mathcal{T}_{\mu_i}, \mathcal{R}_{\psi_i})\}, \qquad (1)$$

where $\mu_i \sim p(\mu)$ and $\psi_i \sim p(\psi)$. State and action spaces remain fixed across the family, reflecting homogeneous changes.

2) The dual challenge: Achieving generative policies for dynamic DCs faces two intertwined challenges:

Reward Form Heterogeneity (C1): A task family requires a shared reward form that aligns with operator intent while adapting to evolving specifications. For instance, cooling efficiency balances energy with operational safety, applicable across scenarios with different thermal loads and SLA bounds ψ. Power capping prioritizes capacity constraints, adapting to both low-density ($\mu_{\min}$) and high-density ($\mu_{\max}$) server rooms with varying SLA limits ($\psi_{\min}, \psi_{\max}$). The challenge is: *how to generate reward forms $\mathcal{R}_\psi$ such that the same structure generalizes across all specifications $(\mu, \psi) \sim p(\mu) \times p(\psi)$?*

Specification-to-Policy Latency (C2): Traditional DRL trains separate agents per specification $\mathcal{M}_i$, requiring hour-level retraining. Specification changes lead to DRL agent performance degradation. In particular, installing additional servers increases heat load and modifies thermal dynamics, creating a dynamics transfer where the actual state transition probability $\mathcal{T}'$ differs from the nominal model $\mathcal{T}$ used during training. Besides, SLA requirements evolve as tenant needs change, yielding a new reward $\mathcal{R}'$ with modified SLA setpoints different from $\mathcal{R}$.

IV. DCoPilot Design

In this section, we present the system design of DCoPilot, which integrates LLM-based symbolic and hypernetwork-based parametric generation of DC control policies.

A. Specification Encoding & Scene Synthesis

A DC operational specification (μ, ψ) defines an operational scenario, which is set according to realistic DC operating ranges. DCoPilot supports interpolation within the trained specification envelope. The scene configuration G describes a DC SimReady scene with each asset paired with a calibrated SimReady model. The dynamics parameter μ captures thermal load variations (e.g., server density, power consumption) that shift transition dynamics $\mathcal{T}_\mu$. The SLA parameter ψ encodes tenant requirements (e.g., server inlet temperature, room humidity) that shape a reward function $\mathcal{R}_\psi$. Each task family corresponds to an operator objective I_{ops} expressed in natural language (e.g., "minimize cooling power consumption while maintaining the air temperature of data hall A").

DCoPilot decomposes specification-to-policy generation into: (i) LLM translates $(G, I_{ops}) \rightarrow \mathcal{R}_\psi(\cdot)$, and (ii) hypernetwork maps $(\mu, \psi) \rightarrow \theta$. To enable generalization, DCoPilot synthesizes diverse training scenarios by sampling specification distributions. Starting from the current calibrated scene G_0, we generate scene variants by modifying asset configurations. Each scene G_i corresponds to a unique dynamics parameter μ_i (e.g., G_1 with $80\,\mathrm{kW}$ load has μ_1, G_2 with $120\,\mathrm{kW}$ has μ_2), yielding $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_m\}$. For SLAs,

we sample $\boldsymbol{\psi} = \{\psi_1, \ldots, \psi_n\}$ with varying bounds (e.g., $T_{\max} \in \{22\,^\circ\mathrm{C}, 25\,^\circ\mathrm{C}, 27\,^\circ\mathrm{C}\}$). The Cartesian product $\boldsymbol{\mu} \otimes \boldsymbol{\psi}$ yields a set of task embeddings $\mathbf{e}$ to define MDP instances as:

$$\mathbf{e} = \{(\mu_i, \psi_j) \mid i = 1, \ldots, m, \ j = 1, \ldots, n\}, \qquad (2)$$

where each (μ_i, ψ_j) pairs scene G_i with SLA ψ_j. By training on this finite set, the hypernetwork learns to interpolate across the continuous parameter space, thereby generating policies for specifications within the sampled range.

B. LLM-Based Symbolic Reward Generation

1) Family reward design: To address **C1**, DCoPilot employs an LLM to synthesize a shared reward form that generalizes across the MDP family $\mathscr{M}$. Figure 5 shows the symbolic reward generation workflow. Unlike piecewise reward design for individual MDPs $\mathcal{M}_i$, family reward synthesis produces a parameterized structure applicable across specification distributions $(\mu, \psi) \sim p(\mu) \times p(\psi)$.

Given scene configuration $G(\mu)$ with environment specification μ and operator objective I_{ops} in natural language, the Reward LLM generates executable reward code. We contextualize I_{ops}, scene G, and DRL training configuration I_{cfg} using prompt template C_{reward} as $Q_{\mathrm{reward}} = C_{\mathrm{reward}}(I_{\mathrm{ops}}, I_{\mathrm{cfg}}, G)$. The Reward LLM produces N candidate reward forms (❶):

$$\{\bar{R}_i\}_{i=1}^N \sim \Theta_{\mathrm{reward}}(Q_{\mathrm{reward}}), \qquad (3)$$

where $\bar{R}_i$ represents the discrete form of a reward function. An example of a reward form is shown in the appendix of the ArXiv version [40]. Each reward candidate combines a list of parameterized SLA setpoints $\boldsymbol{\psi} = \{\psi_1, \psi_2, \ldots, \psi_m\}$. Thus, a family of reward functions can be denoted as:

$$\mathscr{R} = \{R_i \mid R_i(\mathbf{s}|\boldsymbol{\psi})\}, \qquad (4)$$

where the reward function R_i shares the same functional form $\bar{R}$ with others but with a different parameter ψ. The goal is to identify an effective reward form $\bar{R}^*$ that maximizes policy performance across the specification distribution, which will serve as the foundation for subsequent policy training and hypernetwork distillation.

2) Evolutionary optimization: To enable iterative reward refinement, we evaluate candidates on boundary conditions that stress-test generalization across the specification distribution. Rather than testing on random samples, we choose the minimum and maximum values of both environment and SLA parameters, denoted as $\langle \psi_{\min}, \psi_{\max} \rangle$ and $\langle \mu_{\min}, \mu_{\max} \rangle$. The Cartesian product yields boundary task embeddings as:

$$\mathbf{e}_{\mathrm{bound}} = \langle \psi_{\min}, \psi_{\max} \rangle \otimes \langle \mu_{\min}, \mu_{\max} \rangle. \qquad (5)$$

For each candidate $\bar{R}_i$ and boundary condition $(\mu, \psi) \in \mathbf{e}_{\mathrm{bound}}$, we train a DRL policy by maximizing expected return (❷):

$$\pi_{i,\mu,\psi}^* \in \arg\max_\pi \mathbb{E}_{\tau \sim (\pi; G(\mu))} \left[\sum_{t=0}^T \gamma^t R_i(s_t, a_t|\psi) \right]. \qquad (6)$$

Rolling out each trained policy produces trajectories (❸):

$$\pi_{i,\mu,\psi}^*(a|s) \xrightarrow{\text{Rollout on } G(\mu)} \hat{\tau}_i(\mu, \psi), \quad \forall (\mu, \psi) \in \mathbf{e}_{\mathrm{bound}}. \qquad (7)$$

From each trajectory $\hat{\tau}_i(\mu, \psi)$, DCoPilot computes SLA violation costs and objective scores. These metrics are aggregated into the context describing performance at boundary conditions (❹). The top-k reward forms and their performance summaries are appended to the prompt context, enabling the Reward LLM to generate improved candidates in the next iteration. This evolutionary process continues until convergence, selecting $\bar{R}^*$ as the final reward form for policy pool curation.

3) Trajectory pool curation: With the best reward form $\bar{R}^*$ selected, DCoPilot curates a diverse policy pool by training DRL agents across the sampled specification distribution. The set of task vectors for tasks across different environments and rewards is $\mathbf{e} = \boldsymbol{\mu} \otimes \boldsymbol{\psi}$. For each $e = (\mu, \psi) \in \mathbf{e}$, we train a policy with reward $R(\psi)$ under the SimReady environment $G(\mu)$ (❺):

$$\pi_e^* \in \arg\max_\pi \mathbb{E}_{\tau \sim (\pi; G(\mu))} \left[\sum_{t=0}^T \gamma^t R^*(s_t, a_t|\psi) \right], \qquad (8)$$

where $R^* = \bar{R}^*(\cdot|\psi)$ instantiates the shared reward form with SLA parameters ψ, and $G(\mu)$ denotes the scene corresponding to dynamics μ. The key assumption underlying our approach is that converged DRL training establishes a mapping from each MDP $\mathcal{M}_e \in \mathscr{M}$ to a near-optimal policy $\pi_e^* : \mathcal{S} \to \mathrm{Dist}(\mathcal{A})$. Since we parameterize transition dynamics $\mathcal{T}_\mu$ and reward function R_ψ with μ and ψ, each near-optimal policy can be expressed as $\pi_e^*(a|s, \mu_e, \psi_e)$ conditioned on the task embedding $e = (\mu_e, \psi_e)$ for MDP $\mathcal{M}(\mu_e, \psi_e)$. This parameterization enables the hypernetwork to learn the implicit mapping from task embeddings to policy structures. We rollout each trained policy to collect demonstration trajectories (❻):

$$\pi_e^*(a|s) \xrightarrow{\text{Rollout on } G(\mu)} \hat{\tau}_e = \{(s_t, a_t)\}_{t=0}^T. \qquad (9)$$

Aggregating across all embeddings yields the trajectory pool:

$$\mathbf{T} = \{\hat{\tau}_e \mid \hat{\tau}_e, \ \forall e \in \mathbf{e}\}. \qquad (10)$$

Each trajectory $\hat{\tau}_e$ paired with embedding $e = (\mu, \psi)$ provides supervised examples for hypernetwork distillation, encoding the specification-to-behavior mapping across the MDP family.

C. Hypernetwork-Based Parametric Network Generation

1) Meta policy architecture: To address **C2**, DCoPilot employs a hypernetwork that generates policy weights conditioned on specification embeddings. As illustrated in Figure 6, the architecture consists of a hypernetwork H_Θ and a main policy network π_θ. The hypernetwork H_Θ is a multi-layer perceptron that takes task embedding $e = (\mu, \psi)$ as input and outputs the full weight set θ for the main policy. The embedding encoder first projects μ and ψ into fixed-dimensional representations through separate embedding layers, then concatenates them to form the joint embedding $e \in \mathbb{R}^d$. The hypernetwork processes this embedding through several fully-connected layers, producing weight tensors for each layer of the main policy network.

The main policy network π_θ follows a standard actor architecture for continuous control, where it maps observations
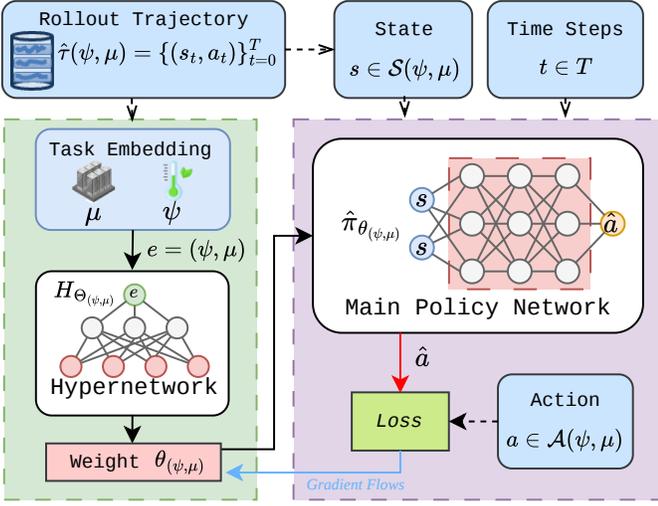
Fig. 6. Overview of the hypernetwork-based policy distillation framework. DCoPilot uses the near-optimal trajectory pool to train the hypernetwork.

$s \in \mathcal{S}$ to action distributions over $\mathcal{A}$. The main policy remains fixed, where only its parameters are generated by H_Θ conditioned on task embeddings:

$$\theta = H_\Theta(e), \quad a \sim \pi_\theta(a|s). \qquad (11)$$

This design enables parameter sharing across the MDP family. Rather than training independent policies for $\mathcal{M}_i \in \mathscr{M}$, DCoPilot learns a single hypernetwork capturing shared control structures for the MDP family $\mathscr{M}$. The hypernetwork implicitly encodes how specifications (μ, ψ) should modulate policy behavior, facilitating zero-shot generalization to new task embeddings.

2) Distillation learning: We cast hypernetwork training as a supervised imitation problem, where for each trajectory τ_i^* the hypernetwork should produce a policy whose action distribution matches the demonstrated actions along the observed states. The hypernetwork training is:

$$\mathcal{L}(\Theta) = -\sum_{i,t} \log \pi_{H_\Theta(e_i)}(a_{i,t} \mid s_{i,t}), \qquad (12)$$

where $e_i \in \mathbf{e}$ encodes the environment and reward parameters (μ_i, ψ_i) corresponding to trajectory τ_i^*. DCoPilot uses the hypernetwork to generalize across different DC environments and reward parameters, enabling zero-shot adaptation of policies for unseen operational specifications.

3) Zero-shot deployment: After distillation, DCoPilot generates the meta policy hypernetwork $H(\Theta)$. For online adaptation to new scenes and SLAs, we first identify the scene feature μ' from monitoring IoT data and set the SLA setpoints ψ'. The hypernetwork $H(\Theta)$ uses new embedding $e' = (\mu', \psi')$ to generate the main policy weight θ' as $\theta' = H(\Theta, e')$.

The generated policy $\pi_{\theta'}$ deploys immediately for online control, eliminating specification-to-policy latency. This zero-shot adaptation does not require environmental interaction, gradient updates, or retraining, addressing the minute-scale control requirements discussed in Section III.

V. EXPERIMENTS

We evaluate DCoPilot on multiple DC cooling control task families, comparing its policies with baselines and ablations to demonstrate the effectiveness of LLM-based symbolic and hypernetwork-based parametric generation.

A. Experimental Setup

1) Task design: We consider two SLAs in today's DC operations: (s1) *temperature*, which maintains the server room temperature within specified bounds, and (s2) *humidity*, which maintains air humidity in some tropical DCs for the contained ITE. Notably, the SLAs only affect indoor facility operations as the serving ITE is contained within the server room. Then, we consider two optimization objectives in DC operations: (o1) *power consumption* of HVAC, which is a major source of DC's energy consumption, and (o2) *water consumption* of chillers, which is also one of the core resource-saving issues.

We combine two SLA targets and two objectives, yielding five task families denoted by (a)-(e). Specifically, we consider the DC as an evolving environment, where the number of installed server units is changing. Task family (a) is applied with the SLA (s1) and the objective (o1) for a single-CRAC server room named *Zone 1*. For task family (a), the observation space captures indoor zone-level states, including the IT workload schedule, zone air temperature, and supply air temperature. The action space consists of continuous control inputs to the CRAC unit, including supply air temperature setpoints and fan airflow rates. Task family (b)-(c) are applied with both (s1) and (s2) and the indoor objective (o1) for a server room named *Zone 2*, including dehumidifiers. For task families (b) and (c), the observation space augments that of task family (a) with relative humidity measurements. Task family (d)-(e) are applied with the SLA (s1) and both (o1) and (o2) for the indoor Zone 1 with outdoor plant facilities denoted as *Indoor+Outdoor Plant*. For task families (d) and (e), the action space is extended to include outdoor plant-level states related to the chilled water system and cooling tower operations.

2) Metrics: For each task family, we use five dynamics parameters $\boldsymbol{\mu}$ by server installations and five SLA parameters $\boldsymbol{\psi}$ by adjusting SLA bounds to synthesize the operational specifications. We set the violation thresholds to $\varepsilon_{s1} = 1\,°\text{C}$ for the temperature SLA (s1) and $\varepsilon_{s2} = 5\,\%\text{RH}$ for the humidity SLA (s2), and compute the violation cost accordingly. We report the time-averaged power usage effectiveness (PUE) for (o1) and water usage effectiveness (WUE) for (o2). The single objective score shown in each cell is the mean over the evaluation horizon. Each control decision in our experiments represents a 15-minute interval.

3) Baselines & ablation settings: We consider four of the existing baseline methods for LLM-generated control policies and one ablation setting. We evaluate baselines and our method on the above task families to exhibit generative policy performance and online adaptation cost. The baselines include (i) *LLM4PID* [16] uses RAG-based LLM to write PID control codes for certain targets. (ii) *LLM-as-Policy*

Method	Control		Zone 1	Zone 2		Indoor + Outdoor Plant	
	Action	**Transfer**	(a) (s1/o1)↓	(b) (s1/o1) ↓	(c) (s1/s2/o1) ↓	(d) (s1/o2) ↓	(e) (s1/o1/o2) ↓
(i) LLM4PID	PID	N/A	0.05/1.75	0.36/1.69	0.36/0.95/1.69	0.34/3.94	0.34/1.64/3.94
(ii) LLM-as-Policy	LLM	N/A	5.90/1.60	3.46/1.67	3.32/3.22/1.66	3.13/3.83	3.27/1.58/3.80
(iii) EvoReward	DRL	No	1.18/1.72	3.09/1.64	0.99/1.69/1.72	8.78/3.96	4.05/1.76/4.21
ABL (w/o bound)	Meta	Env+Rew	1.42/1.73	0.82/1.72	0.44/0.80/1.76	0.07/3.86	0.32/1.58/3.77
DCoPilot	Meta	Env+Rew	**0.01/1.63**	**0.02/1.68**	**0.01/0.00/1.69**	**0.18/3.80**	**0.19/1.59/3.80**

[15] uses historical data as in-context examples with few-shot prompting for direct industrial control. (iii) *EvoReward* [19] exploits the LLMs to perform evolutionary optimization over reward code and to train the corresponding policy via reinforcement learning. We also include one ablation setting to show the effects of family reward code generation, which is *ABL (w/o bound)* removes the boundary trajectory information during the reward evolution. We conduct the experiments with `GPT-3.5-Turbo` and fix the LLM sampling parameters (temperature=0.7, $N = 5$).

B. Generative Policy Evaluation

1) Performance across generative approaches: We evaluate how different generative methods handle within-family specification variations by training on one representative task and testing across all specifications in each family. Table I presents the average violation costs and objective scores, where lower values indicate better performance. DCoPilot achieves the best performance across all task families, maintaining violation costs below $0.2\,°C$ while baseline methods range from $0.77\,°C$ to $8.78\,°C$. This significant improvement demonstrates the power of combining two forms of generative AI, i.e., LLMs for semantic understanding and hypernetworks for parametric policy generation. The LLM translates natural language specifications into appropriate reward structures, while the hypernetwork generates policy weights. Direct LLM generation methods reveal fundamental limitations when applied to control tasks. LLM-as-Policy produces violation costs from $3.13\,°C$ to $5.90\,°C$ because token-by-token generation cannot maintain control consistency across time steps. Each action is generated independently, causing unstable control behaviors. LLM4PID performs better on constraints but poorly on objectives, achieving scores of 1.64/3.94 versus DCoPilot's 1.59/3.80. While LLMs can tune PID parameters, they cannot overcome the inherent limitations of classical control for multi-objective optimization. Task-specific DRL methods cannot generalize across specifications. EvoReward shows catastrophic performance on complex tasks, reaching $8.78\,°C$ violations in task family (d), because each agent learns fixed behaviors tied to its training configuration. These methods cannot adapt to new specifications without complete retraining. Our ablation study confirms the importance of guided reward generation. Removing boundary trajectory guidance increases violations by 5 to 70 times across different families.
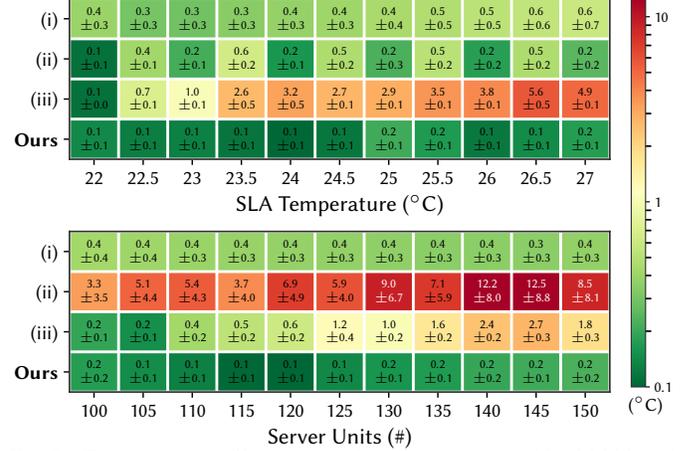

Fig. 7. The temperature SLA error of the policies generated by LLM-based generative policies (i)-(iii) and DCoPilot.

The boundary trajectories help the LLM understand safety boundaries when generating rewards, transforming abstract SLA requirements into concrete optimization objectives that the hypernetwork can learn effectively.

2) Robust within-family generalization: Figure 7 visualizes why DCoPilot succeeds where task-specific methods fail across two critical generalization dimensions, systematically evaluating all four approaches on task family (a) across varying SLA temperature bounds ($22\,°C$-$27\,°C$) and server densities (100-150 units). LLM4PID maintains good performance in SLA generalization at $0.43\,°C$ mean absolute error (MAE) and similarly in environment generalization at $0.38\,°C$ MAE. While generating PID codes for DC control can fulfill SLA constraints, it is incapable of objective optimization as shown in Table I. LLM-as-Policy achieves $0.33\,°C$ error across SLA variations through retrieval from its demonstration database. However, it catastrophically fails in environment generalization with $7.24\,°C$ mean error and 100% violations at 140+ server units. This RAG-based approach cannot find relevant examples for new specifications outside its history distribution, and thus generates risky actions in unseen environments. In contrast, DCoPilot maintains consistent sub-degree temperature errors across all tested configurations, achieving mean errors of $0.43\,°C$ for SLA generalization and $0.38\,°C$ for environment generalization with standard deviations below $1\,°C$. This consistent performance stems from the hypernetwork architecture, which learns the mapping from specifications to
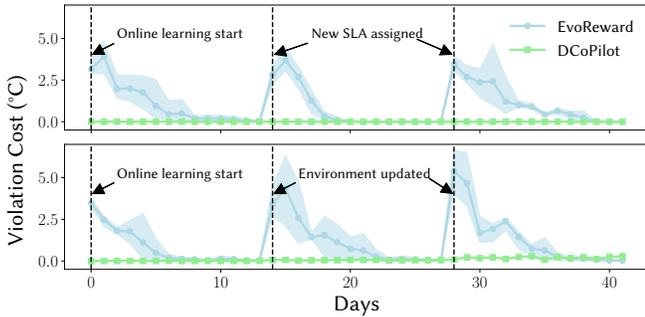
Fig. 8. *Day-level* online adaptation violation cost of assigning new SLAs and updating environments in task family (a) by EvoReward and DCoPilot.

Fig. 9. (Left) Temperature MAE and (right) loss comparison between Hypernetwork and CPN on task family (a).

appropriate policy weights rather than fixed behaviors tied to specific training configurations. This architectural difference explains the performance gap in Table I.

3) Day-level performance under online specifications: To evaluate how different generative methods handle realistic operational changes, we simulate a 40-day DC operation scenario with specification changes occurring at days 14 and 28. These changes represent common operational events, i.e., initiating online learning for initial deployment, assigning new SLA requirements, and updating server infrastructure. Figure 8 presents the accumulated violation costs during these transitions. EvoReward exhibits severe spikes in violation at each specification change, with costs ranging up to $4\,°C$ when new SLAs are assigned, or the environment is updated. These spikes occur because EvoReward must retrain entirely new agents for each specification, leaving the system effectively uncontrolled during a 10-day online training period. In contrast, DCoPilot maintains near-zero violations throughout all specification changes, with costs remaining below $0.1\,°C$ even during transitions. This stability stems from the hypernetwork's instant policy generation without any online adaptation. When SLAs or server capacity change (days 14 and 28), DCoPilot immediately produces updated policy weights with the corresponding task embeddings.

C. Minute-Level Response with Hypernetwork

1) Policy-Parameter Generation vs. Action-Level Conditioning: A natural alternative to hypernetwork-based policy generation is to directly condition the policy on the control specification, i.e., learning a conditional policy of the form $\pi(a \mid s, e)$ as in conditional policy networks (CPNs). Figure 9 shows that using hypernetworks achieves a lower temperature MAE at 0.31 compared to CPN at 1.22 on interpolated but unseen specifications. While CPN exhibits a faster decrease in training loss, it suffers from consistently higher validation loss. This behavior arises from its inductive bias, where CPN performs action-level interpolation in the joint state–specification space and implicitly assumes smooth transitions of the optimal action with respect to specification changes. However, in DC control, even interpolated specification changes can induce substantial shifts in the underlying state distribution, rendering such interpolation unreliable. In contrast, the hypernetwork explicitly models how control specifications transform policies
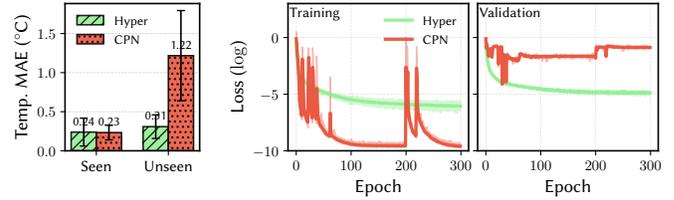
by learning a mapping $e \mapsto \theta_e$, followed by execution via $\pi_{\theta_e}(a \mid s)$. By performing interpolation at the parameter level, the hypernetwork generates coherent task-specific policies.

2) Comparison with Meta-Learning Methods: We further compare DCoPilot against two prominent meta-learning baselines, MAML and PEARL, under identical training conditions using the same LLM-generated reward form and specification distribution. We evaluate zero-shot transfer across four consecutive specification changes (100, 115, 135, and 185 servers) over a 24-hour control horizon, deploying all methods without online fine-tuning to simulate realistic scenarios requiring immediate policy adaptation to workload or hardware changes.

Figure 10 shows simulation results for MAML, PEARL, and the hypernetwork across three dimensions, including IT power consumption, total power consumption, and zone air temperature. Throughout four specification transitions, DCoPilot demonstrates superior control stability across all metrics. For total power efficiency, DCoPilot maintains consistently lower power consumption, indicating more effective cooling coordination. Most notably, in temperature regulation, DCoPilot maintains the DC temperature stably within the SLA-acceptable range around the $22\,°C$ setpoint throughout all specification transitions, whereas both MAML and PEARL exhibit larger temperature deviations, particularly during workload changes. The superior performance comes from the hypernetwork directly generating task-specific parameters via explicit specification-to-policy mapping, unlike MAML's gradient-based adaptation and PEARL's probabilistic context inference. The generative control paradigm removes adaptation latency and performance issues seen in meta-learning when specifications exceed training coverage.

D. Role of LLM in Meta Policy Learning

1) Family reward form enables hypernetwork convergence: We assess LLM-generated family reward forms for hypernetwork training by comparing DCoPilot's unified reward generation to a piecewise reward generation approach, where the LLM creates separate reward functions per specification, trains DRL agents on each, and then distills these agents into a hypernetwork. Figure 11 presents the training loss on task family (a). DCoPilot with unified reward generation achieves stable convergence in hypernetwork training, demonstrating effective interpolation between different specifications. The piecewise reward generation baseline fails to converge, exhibiting 100x higher loss throughout training. The LLM generates unified reward forms covering all specifications, which ensures all trained policies share the same MDP family.
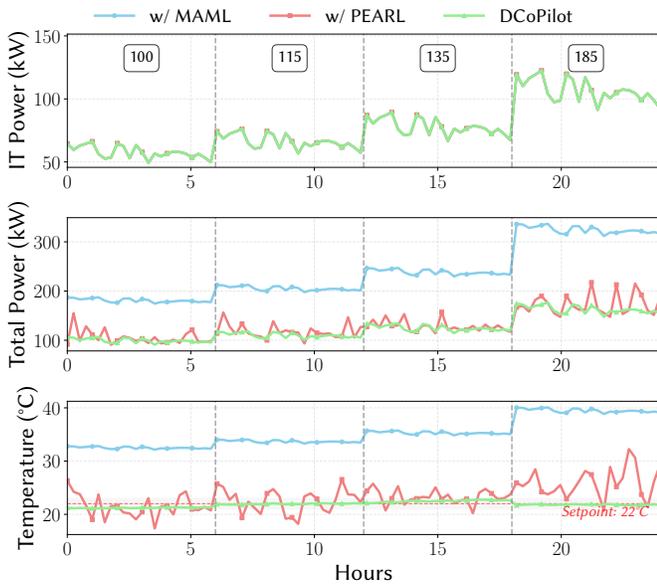
Fig. 10. *Minute-level* online adaptation violation cost of new specifications (e.g., server units=100, 115, 135, 185) in task family (a) by MAML, PEARL, and hypernetwork for parametric policy weight generation of DCoPilot.
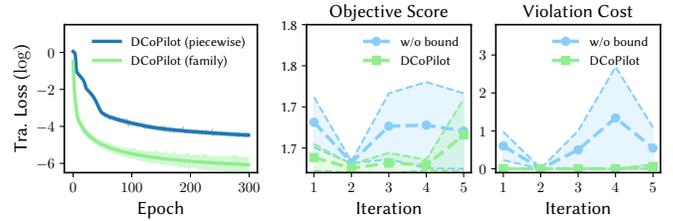

Fig. 11. (Left) Training loss convergence comparing unified reward generation against piecewise reward generation. (Middle, Right) Evolution of objective score and violation cost across reward refinement iterations. All experiments conducted on task family (a).

2) Boundary trajectories stabilize reward evolution: We evaluate the effectiveness of boundary trajectory guidance during LLM-based reward evolution by comparing DCoPilot against the ablation baseline without boundary information. Reward quality at each iteration is measured by the average violation cost and objective score of the best candidate, evaluated across five different LLM sampling seeds. Figure 11 presents the evolution trends on task family (a). DCoPilot demonstrates rapid convergence to near-zero violation costs within the first few iterations and maintains stable performance throughout subsequent rounds. In contrast, the baseline without boundary specifications trajectories exhibits fluctuating violation costs, occasionally producing reward candidates that cannot adequately encode safety constraints. Both methods achieve comparable objective scores, but DCoPilot's consistent constraint satisfaction demonstrates more reliable reward generation. This stability comes from the boundary trajectory evaluation strategy, where testing reward candidates on extreme specifications provides comprehensive stress-testing.

VI. DISCUSSION

DCoPilot's dual-generative architecture addresses a core efficiency challenge in AIDC operations. It shifts all training overhead offline through LLM-based automation and instantly generates policy online. A quantitative analysis of the offline computational cost is shown in the appendix of the ArXiv version [40]. Although it shows strong zero-shot generalization across diverse control tasks, four key challenges remain. First, bridging the gap between simulated and real DC dynamics is difficult due to mismatches in airflow, heat transfer, and actuator latency [41]. Hypernetworks trained on synthetic data may falter with sensor noise or unmodeled system couplings in real settings. Accurate SimReady digital twins need continuous recalibration with evolving hardware and workloads [42].

Future work should focus on domain adaptation methods to ensure robust policy transfer to live environments. Second, neither the LLM-based reward generator nor the hypernetwork offers formal convergence guarantees under non-convex optimization. LLM outputs are generated stochastically rather than optimized by gradient descent, causing variability in reward definitions [43]. The hypernetwork, trained over a distribution of tasks instead of a single objective, may exhibit unstable convergence [44]. Future work will explore theoretical frameworks for the convergence. Third, this paper only considers homogeneous specification changes that can be parameterized, but real-world DC evolution often involves heterogeneous changes to the facility topology [45]. Addressing this limitation needs a hierarchical framework, where an LLM provides expert-guided recommendations for complex heterogeneous changes, while low-level controllers handle the detailed setpoints. Future work will also explore heterogeneous topological changes via modular hypernetworks, where each generates policy parameters for a specific subsystem. Moreover, our approach only provides soft constraint adherence through reward shaping at training time rather than worst-case guarantees at serving time. Incorporating explicit serving-time safety layers [46] is an important direction for future work.

VII. CONCLUSION

This paper presents DCoPilot, a hybrid framework that synergizes LLM-based symbolic reward generation with hypernetwork-based parametric policy generation to address specification-to-policy latency in dynamic DC control. This work establishes a new paradigm for generative CPS control, where semantic understanding from pre-trained language models combines with parametric precision from neural policy generators to enable automated and adaptive operations in mission-critical, evolving industrial environments.

REFERENCES

[1] Dell'Oro Group, "Data center capex to grow at 21 percent cagr through 2029," https://www.delloro.com/news/data-center-capex-to-grow-at-21-percent-cagr-through-2029/, 2025.

[2] P. Carton, A. Sbarra, L. Williams, and T. Machado, "Uptime institute ai infrastructure survey 2025: Summary," Uptime Institute, Tech. Rep., 2025. [Online]. Available: https://nextdc.com/hubfs/Uptime%20Institute%20AI%20Infrastructure%20Survey%202025.pdf

[3] McKinsey & Company. (2024, October) Ai power: Expanding data center capacity to meet growing demand. [Online]. Available: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/ai-power-expanding-data-center-capacity-to-meet-growing-demand

[4] Google Cloud. (2025) Mitigating power and thermal fluctuations in ml infrastructure. [Online]. Available: https://cloud.google.com/blog/topics/systems/mitigating-power-and-thermal-fluctuations-in-ml-infrastructure

[5] Google DeepMind, "Safety-first ai for autonomous data centre cooling and industrial control," https://deepmind.google/discover/blog/safety-first-ai-for-autonomous-data-centre-cooling-and-industrial-control/, 2018.

[6] E. Wustenhoff and S. BluePrints, "Service level management in the data center," Sun BluePrints Online, vol. 1, pp. 2–13, 2002.

[7] N. Lazic, C. Boutilier, T. Lu, E. Wong, B. Roy, M. Ryu, and G. Imwalle, "Data center cooling using model-predictive control," Advances in Neural Information Processing Systems, vol. 31, 2018.

[8] S. Moss. (2023, 10) Equinix blames singapore outage on third party contractor. [Online]. Available: https://www.datacenterdynamics.com/en/news/equinix-blames-singapore-outage-on-third-party-contractor/

[9] B. Durand-Estebe, C. Le Bot, J. N. Mancos, and E. Arquis, "Data center optimization using pid regulation in cfd simulations," Energy and Buildings, vol. 66, pp. 154–164, 2013.

[10] M. H. Beitelmal and C. D. Patel, "Model-based approach for optimizing a data center centralized cooling system," Hewlett-Packard (HP) Lab Technical Report, 2006.

[11] T. Moriyama, G. De Magistris, M. Tatsubori, T.-H. Pham, A. Munawar, and R. Tachibana, "Reinforcement learning testbed for power-consumption optimization," in Methods and Applications for Modeling and Simulation of Complex Systems: 18th Asia Simulation Conference, AsiaSim 2018, Kyoto, Japan, October 27–29, 2018, Proceedings 18. Springer, 2018, pp. 45–59.

[12] Y. Li, X. Hu, Y. Zhuang, Z. Gao, P. Zhang, and N. El-Sheimy, "Deep reinforcement learning (drl): Another perspective for unsupervised wireless localization," ieee internet of things journal, vol. 7, no. 7, pp. 6279–6287, 2019.

[13] K. S. H. Ong, W. Wang, D. Niyato, and T. Friedrichs, "Deep-reinforcement-learning-based predictive maintenance model for effective resource management in industrial iot," IEEE Internet of Things Journal, vol. 9, no. 7, pp. 5173–5188, 2021.

[14] R. Wang, Z. Cao, X. Zhou, Y. Wen, and R. Tan, "Phyllis: Physics-informed lifelong reinforcement learning for data center cooling control," in Proceedings of the 14th ACM International Conference on Future Energy Systems, 2023, pp. 114–126.

[15] L. Song, C. Zhang, L. Zhao, and J. Bian, "Pre-trained large language models for industrial control," arXiv preprint arXiv:2308.03028, 2023.

[16] H. Koziolek, S. Grüner, R. Hark, V. Ashiwal, S. Linsbauer, and N. Eskandani, "Llm-based and retrieval-augmented control code generation," in Proc. 1st Int. Workshop on Large Language Models for Coffice (LLM4Code) at ICSE, vol. 2024, 2024.

[17] M. Fakih, R. Dharmaji, Y. Moghaddas, G. Quiros, O. Ogundare, and M. A. Al Faruque, "Llm4plc: Harnessing large language models for verifiable programming of plcs in industrial control systems," in Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice, 2024, pp. 192–203.

[18] J. Song, Z. Zhou, J. Liu, C. Fang, Z. Shu, and L. Ma, "Self-refined large language model as automated reward function designer for deep reinforcement learning in robotics," arXiv preprint arXiv:2309.06687, 2023.

[19] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," in The Twelfth International Conference on Learning Representations, 2024. [Online]. Available: https://openreview.net/forum?id=IEduRUO55F

[20] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," arXiv preprint arXiv:1609.09106, 2016.

[21] NVIDIA Corporation, "Simready assets," NVIDIA Omniverse Developer Portal, 2025. [Online]. Available: https://developer.nvidia.com/omniverse/simready-assets

[22] S. Pasricha, "Mosaic: A multi-objective optimization framework for sustainable datacenter management," Nov 2023.

[23] Y. Ran, H. Hu, X. Zhou, and Y. Wen, "Deepee: Joint optimization of job scheduling and cooling control for data center energy efficiency using deep reinforcement learning," in 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019, pp. 645–655.

[24] R. Wang, X. Zhou, Y. Wen, and R. Tan, "Joint it-facility optimization for green data centers via deep reinforcement learning," IEEE Network, vol. 35, no. 5, pp. 51–57, 2021.

[25] S. Sarkar, A. Naug, A. Guillen, R. Luna, V. Gundecha, A. R. Babu, and S. Mousavi, "Sustainability of data center digital twins with reinforcement learning," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 21, 2024, pp. 23 832–23 834.

[26] X. Zhan, "Data center cooling system optimization using offline reinforcement learning," 2025.

[27] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in International conference on machine learning. PMLR, 2017, pp. 1126–1135.

[28] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in International conference on machine learning. PMLR, 2019, pp. 5331–5340.

[29] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," arXiv preprint arXiv:1606.04671, 2016.

[30] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," Advances in neural information processing systems, vol. 32, 2019.

[31] S. Cho, D. Kim, J. Lee, and S. Hong, "Meta-controller: Few-shot imitation of unseen embodiments and tasks in continuous control," Advances in Neural Information Processing Systems, vol. 37, pp. 134 250–134 286, 2024.

[32] Y. Meng, Z. Bing, X. Yao, K. Chen, K. Huang, Y. Gao, F. Sun, and A. Knoll, "Preserving and combining knowledge in robotic lifelong reinforcement learning," Nature Machine Intelligence, pp. 1–14, 2025.

[33] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," Advances in neural information processing systems, vol. 33, pp. 6840–6851, 2020.

[34] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song, "Diffusion policy: Visuomotor policy learning via action diffusion," The International Journal of Robotics Research, vol. 44, no. 10-11, pp. 1684–1704, 2025.

[35] S. Rezaei-Shoshtari, C. Morissette, F. R. Hogan, G. Dudek, and D. Meger, "Hypernetworks for zero-shot transfer in reinforcement learning," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, no. 8, 2023, pp. 9579–9587.

[36] Y. J. Ma, W. Liang, H.-J. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman, "Dreureka: Language model guided sim-to-real transfer," arXiv preprint arXiv:2406.01967, 2024.

[37] R. Wang, Q. Zhang, J. Jia, Y. Wen, and S. Kennedy, "Toward multiphysics-informed machine learning for sustainable data center operations: Intelligence evolution with deployable solutions for computing infrastructure," IEEE Energy Sustainability Magazine, vol. 2, no. 1, pp. 69–80, 2026.

[38] Y. Li, Y. Wen, D. Tao, and K. Guan, "Transforming cooling optimization for green data center via deep reinforcement learning," IEEE transactions on cybernetics, vol. 50, no. 5, pp. 2002–2013, 2019.

[39] D. Van Le, Y. Liu, R. Wang, R. Tan, Y.-W. Wong, and Y. Wen, "Control of air free-cooled data centers in tropics via deep reinforcement learning," in Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation, 2019, pp. 306–315.

[40] M. Li, R. Wang, R. Tan, and Y. Wen, "Dcopilot: Generative ai-empowered policy adaptation for dynamic data center operations," arXiv preprint arXiv:2602.02137, 2026.

[41] Z. Cao, M. Li, F. Lin, J. Jia, Y. Wen, J. Yin, and S. See, "Transforming future data center operations and management via physical ai," arXiv preprint arXiv:2504.04982, 2025.

[42] C. Vering, S. Borges, D. Coakley, H. Krützfeldt, P. Mehrfeld, and D. Müller, "Digital twin design with on-line calibration for hvac systems in buildings," in *Building Simulation 2021*, vol. 17. IBPSA, 2021, pp. 2938–2945.

[43] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. Ruiz, A. Mehrabian *et al.*, "Alphaevolve: A coding agent for scientific and algorithmic discovery," *arXiv preprint arXiv:2506.13131*, 2025.

[44] E. Sarafian, S. Keynan, and S. Kraus, "Recomposing the reinforcement learning building blocks with hypernetworks," in *International Confer-ence on Machine Learning*. Pmlr, 2021, pp. 9301–9312.

[45] L. Poutievski, O. Mashayekhi, J. Ong, A. Singh, M. Tariq, R. Wang, J. Zhang, V. Beauregard, P. Conner, S. Gribble *et al.*, "Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 66–85.

[46] R. Wang, Z. Cao, X. Zhou, Y. Wen, and R. Tan, "Green data center cooling control via physics-guided safe reinforcement learning," *ACM Transactions on Cyber-Physical Systems*, vol. 8, no. 2, pp. 1–26, 2024.