2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

AURA: Autonomous Upskilling with Retrieval-Augmented Agents

Alvin Zhu[1*], Yusuke Tanaka[2*], Andrew Goldberg[3], Dennis Hong[2]

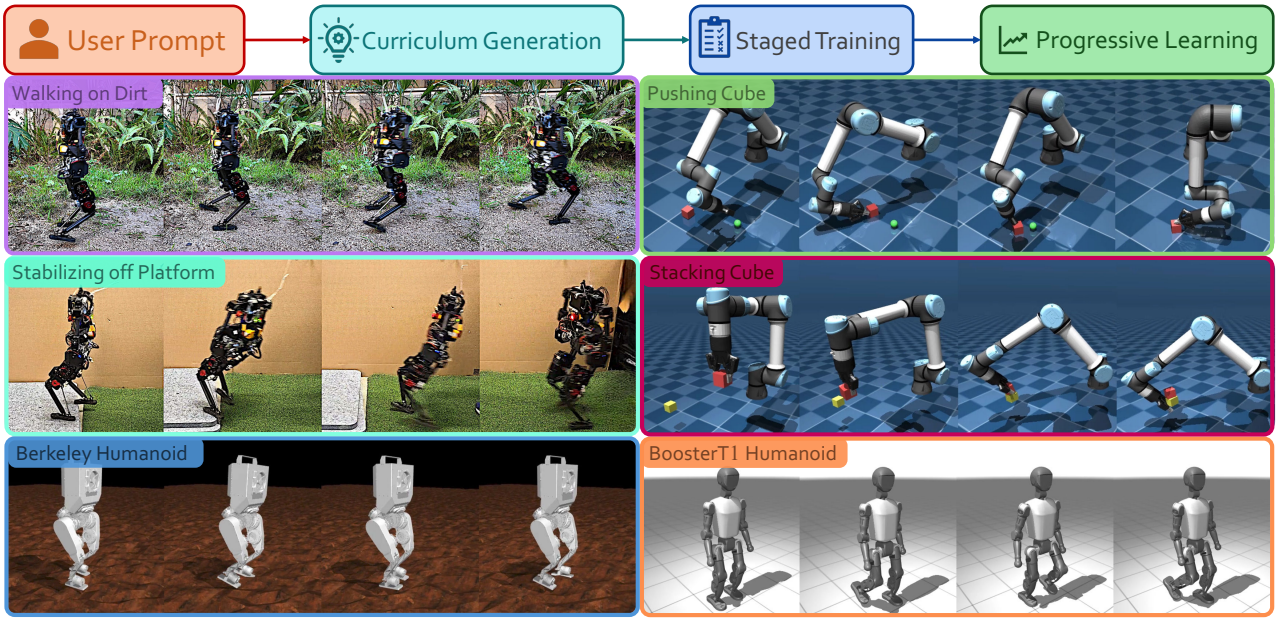

Fig. 1: AURA-trained policies deployed successfully on custom humanoid hardware and in simulation for locomotion and manipulation tasks.

***Abstract*— Designing reinforcement learning curricula for agile robots traditionally requires extensive manual tuning of reward functions, environment randomizations, and training configurations. We introduce AURA (Autonomous Upskilling with Retrieval-Augmented Agents), a schema-validated curriculum reinforcement learning (RL) framework that leverages Large Language Models (LLMs) as autonomous designers of multi-stage curricula. AURA transforms user prompts into YAML workflows that encode full reward functions, domain randomization strategies, and training configurations. All files are statically validated before any GPU time is used, ensuring efficient and reliable execution. A retrieval-augmented feedback loop allows specialized LLM agents to design, execute, and refine curriculum stages based on prior training results stored in a vector database, enabling continual improvement over time. Quantitative experiments show that AURA consistently outperforms LLM-guided baselines in generation success rate, humanoid locomotion, and manipulation tasks. Ablation studies highlight the importance of schema validation and retrieval for curriculum quality. AURA successfully trains end-to-end policies directly from user prompts and deploys them zero-shot on a custom humanoid robot in multiple environments —capabilities that did not exist previously with manually designed controllers. By abstracting the complexity of curriculum design, AURA enables scalable and adaptive policy learning pipelines that would be complex to construct by hand. aura-research.org**

[1]A. Zhu is with Department of Computer Science and Electrical Engineering, [2]Y. Tanaka and D. Hong are with Department of Mechanical and Aerospace Engineering, UCLA, Los Angeles, CA, USA. [3]A. Goldberg is with Department Electrical Engineering and Computer Science, UC Berkeley, Berkeley, CA, USA. {alvin.zhu, yusuketanaka, dennishong}@g.ucla.edu. apgoldberg@berkeley.edu. * denotes equal contribution.

I. Introduction

Curriculum reinforcement learning (RL) [1–4] enables robots to master complex skills by decomposing tasks into progressively harder subtasks [5]. This approach has proven to be effective in domains like agile locomotion [6–9], where single-stage learning can struggle with sparse rewards or high-dimensional exploration spaces. By splitting the learning process, agents are guided through increasingly challenging environments, improving sample efficiency and convergence.

However, designing effective multi-stage curricula remains a significant bottleneck [10, 11]. Multi-stage curricula require coordinated changes across reward shaping, randomization, simulation fidelity, and optimization; as the number of stages increase, designing transitions, verifying stability, and tuning parameters scales combinatorially (the "curse of dimensionality" [12]) and becomes brittle to human error and heuristic bias [13, 14]. Moreover, failures at any stage often derail the entire training process, making automation critical for scaling curriculum RL.

Large language models (LLMs) offer a compelling alternative to optimization-based approaches [15]. Their high-level reasoning and generalization capabilities have demonstrated success in robotic planning, perception, and code generation [16–21]. While LLMs are not directly deployable on real-time control loops due to latency constraints [22], LLM's

have shown potential in *designing* training schemes for real-time robot control policies [23–25]. However, current LLM-based RL pipelines inefficiently use computation on launching parallel environments to deal with malformed generations and fail to learn from experience across tasks.

What is missing is a principled framework for transforming high-level prompts into reliable, executable RL training pipelines that improve with experience. Such a system should (1) ensure syntactic and semantic validity before execution, (2) use past experience to optimize future performance, and (3) modularize the pipeline into verifiable components [26]. Without these capabilities, LLMs are limited to more fragile, trial-and-error reward design rather than scalable curriculum generation.

AURA (*Autonomous Upskilling with Retrieval-Augmented Agents*) addresses these bottlenecks with three key ideas. (i) A *typed YAML schema* captures curricula, reward functions, randomization, and training hyperparameters; structured LLM outputs are statically validated before a single GPU cycle is spent. (ii) A *team of specialized LLM agents* works in collaboration to enable modular, schema-compliant curriculum generation. (iii) A *Retrieval-augmented generation (RAG)* module with a *vector database (VDB)* stores prior task specifications, curricula, and rollout evaluations. This database supports experience-informed generation by enabling agents to condition on successful prior workflows, improving both accuracy and generalization.

We evaluate AURA on a suite of humanoid locomotion and deploy the resulting policies zero-shot on a custom, kid-sized humanoid. Compared to curriculum RL baselines and recent LLM-crafted reward pipelines, AURA is capable of generating context-rich, high-dimensional curricula that produce successful zero-shot policies deployable on real humanoid hardware. Our contributions are:

1) **AURA**: A fully agentic, retrieval-augmented framework that turns a natural-language prompt into a hardware-deployable controller policy.
2) **Curriculum compiler, schema-validation**: A typed YAML schema that provides an LLM-friendly interface for defining reward terms, domain randomizations, and training configurations. Static validation of the YAMLs enables descriptive error messages for generation retries, without wasting compute on failed environment launches.
3) **Experience-aware, self-improving, curriculum generation agent**: Specialized LLM agents query a vector database of past runs and evaluations, select relevant training files, generate high-level curricula, and refine them into executable multi-stage specifications. This feedback-driven loop promotes curriculum quality and training stability over many iterations.
4) **Zero-shot deployment**: Experiments showing AURA's policies transfer zero-shot to a custom kid-sized humanoid, Berkeley Humanoid, Booster T1, [27], and manipulators, showing end-to-end prompt-to-policy ability.

II. Related Work

A. Large Language Models in Robotics

Early efforts to link natural language to robot control and task-planning frameworks translate natural-language commands into symbolic action graphs, as demonstrated by SayCan and Code-as-Policies [17, 18]. Recent Vision–Language–Action paradigms, such as RT-1/2, $\pi 0$, Octo, and OTTER, [16, 28–31], demonstrate that pretraining on large-scale robot data yields policies that generalize across many robotic skills. Further work has extended this to cross-domain generalization, deformable object manipulation, and long-horizon household tasks, exemplified by Gemini Robotics and embodied LLMs [21, 22]. Other lines of work have explored LLM-driven planning and language-augmented reasoning for robotic control pipelines [19–21, 32–35]. Despite their semantic flexibility, these methods continue to rely on hand-tuned, low-level controllers and standard actuation assumptions, with LLMs typically external to the closed control loop [32, 33]. RoboGen addresses these limitations through decomposing long-horizon tasks into skills, which are tuned or trained by LLMs [36].

B. Curriculum Reinforcement Learning

Reinforcement learning often struggles with sparse rewards and long-horizon problems, where meaningful feedback is rare or delayed over many timesteps [37]. Curriculum RL addresses sparse-reward and long-horizon problems by exposing agents to a progression of gradually more difficult tasks [1, 5, 14]. Surveys and benchmarking studies note that practical curriculum design remains heuristic and sensitive to context-specific particulars [14]. Reverse-curriculum approaches [38, 39] employ a staged training process that progressively increases the task's difficulty from previous successful demonstrations, thereby enhancing exploration and sample efficiency. Domain-randomization and curriculum-based RL have enabled real-world legged locomotion and manipulation in challenging environments [2–4, 6–9, 40–44].

C. LLM-guided RL and Curricula

Emerging systems such as CurricuLLM [24] and Eureka [23, 45] utilize LLMs to generate reward functions for RL training. Eureka uses evolutionary search and prompts LLMs in parallel to directly generate reward code for various tasks in simulation. DrEureka extends Eureka to generate domain randomizations and consider hardware safety, demonstrating real-world locomotion tasks on a Unitree Go1 quadruped robot [25]. CurricuLLM [24] similarly uses evolutionary search and directly generates code, but focuses on curriculum generation and humanoid locomotion, showing real-world success on the Berkeley Humanoid [27].

These frameworks require sampling a large number of training runs to obtain a viable or high-performing policy, relying on LLM stochasticity and brute-force search. RAG and VDBs are approaches to enhance the LLM knowledge beyond what the LLM has been trained on [46], which AURA utilizes to construct effective curricula.

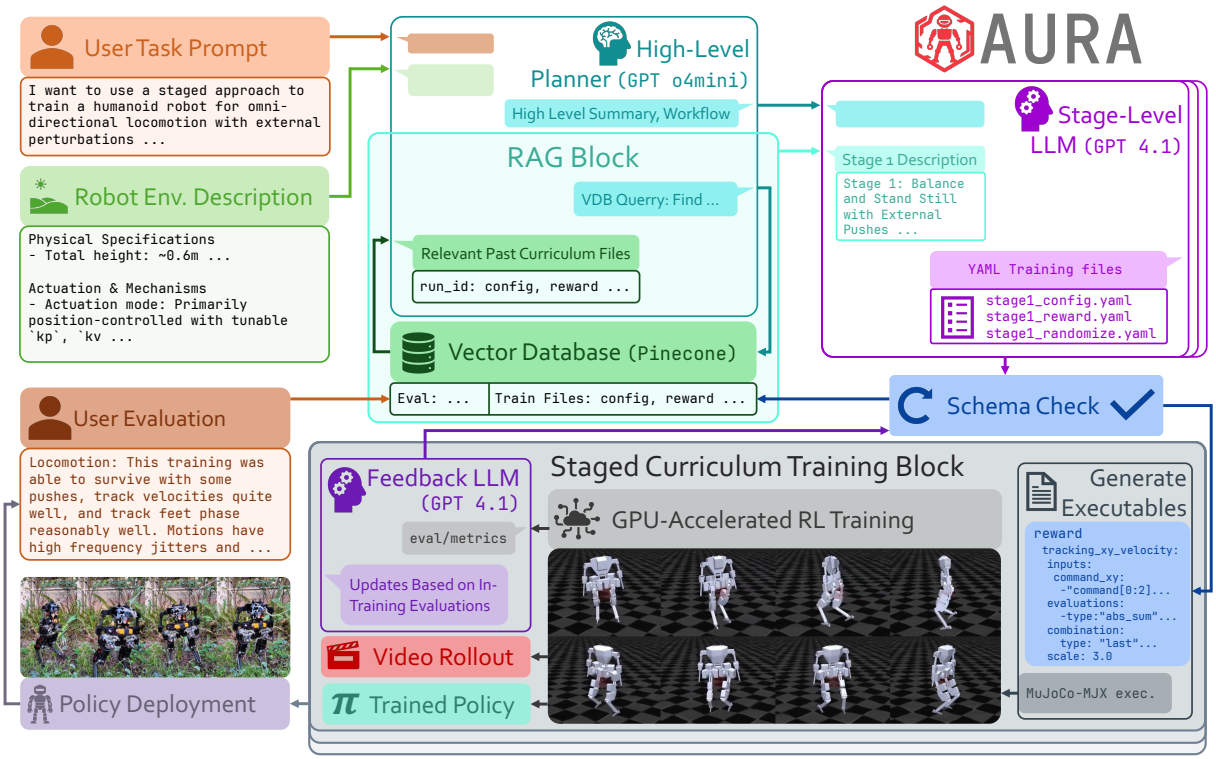

Fig. 2: An overview of the AURA curriculum generation and policy training framework.

Positioning of this work: AURA unifies curriculum generation, domain randomization, and training configuration within a schema-validated, retrieval-augmented loop that learns from prior runs. This yields consistent, deployable specifications with higher first-attempt launch rates and removes the need for post hoc sampling or manual curation.

III. Methods

A. Problem Setup and Curriculum Formalism

We define a curriculum $\mathcal{C}$ as an ordered sequence of K training stages, $\mathcal{C} = \{\xi_1, \ldots, \xi_K\}$, where each stage $\xi_k = \langle \Phi_k, \rho_k, \Theta_k, \kappa_k \rangle$ is comprised of a reward Φ_k, domain-randomization distributions ρ_k over environment parameters ψ, a training configuration Θ_k, and a promotion criterion $\kappa_k(\pi)$ that governs advancement. The curriculum stages are modeled as an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}(\cdot \mid s, a; \psi), R_\psi, \gamma \rangle$, [47, 48] with ψ denoting randomized physics properties [49, 50]. AURA generates $(\Phi_k, \rho_k, \Theta_k)$ for each stage as schema-compliant YAML files that are statically validated and compiled into an MJX-based RL pipeline [51, 52]. We use this notation to describe generation, validation, training, and iteration.

B. Inputs to Curriculum Generation

To instantiate a curriculum from a prompt, AURA consumes four inputs: (i) a natural language task description specifying the desired behavior; (ii) a structured robot specification: joint names, semantic labels (e.g., "left foot contact"), available sensors, and actuator limits; (iii) an MJCF simulation model encoding robot kinematics, dynamics, and contact geometry; and (iv) a Python MJX environment defining observations, actions, physics parameters, and episode termination conditions.

C. Retrieval-Augmented High-Level Planning

AURA begins by leveraging LLMs and a vector database of prior curricula and outcomes to generate a high-level plan. The user-provided task description and robot specification are first passed into a *query LLM*, which synthesizes them into a structured retrieval query. This query is then encoded with OpenAI's `text-embedding-3-large` [53] and used to perform cosine-similarity search over the VDB (Pinecone). From the top-3 most relevant prior curricula, the database returns reward, randomization, and training files, along with user feedback. A *selector LLM* then chooses the single most relevant example from this set, based on both user feedback and curricula relevance to the current task; this example is included in the high-level planner's context for generating the new curricula.

Conditioned on this selected example and the user task specification, a high-level planner decides the number of stages and produces a natural-language plan for each one. The plan describes ideas for reward components, domain randomizations, and stage-specific training hyperparameters (e.g., terrain type, disturbance level, PPO hyperparameters, etc.).

The VDB enables two modes of AURA. (1) AURA Blind where the VDB is initialized as empty for the first iteration and AURA must design a reward from scratch, and (2)

AURA Tune where the VDB is initialized with human-expert rewards from a related task and must *tune* that reward for the current domain.

D. Stage-Level Specification and YAML Generation

Each stage plan from the high-level planner is given to an independent stage-level generator which translates the stage plan into structured YAML files. The stage-level generator takes as input: (i) the user task description; (ii) the natural-language stage description produced by the high-level planner; (iii) the available reward function library; (iv) state variables and sensor signals exposed by the robot environment; (v) the selected prior curriculum example retrieved from the VDB (chosen from the top-3 retrieval set); (vi) context from previously generated stages in the current workflow; and (vii) the reward/schema format specifications.

Each stage description ξ_k is translated into three human-readable, schema-compliant YAML files: *Reward YAML* (Φ_k) defines the differentiable terms, coefficients, and aggregation logic with explicit sensor/state dependencies; *Randomization YAML* (ρ_k) defines the target parameters, distributions, and activation conditions; and *Training Config YAML* (Θ_k) defines the PPO hyperparameters, episode budgets, checkpointing cadence, normalizations, etc.. These files provide the abstract specifications needed for execution without requiring direct generation of JAX code.

E. Schema Validation and Compilation

Directly emitting JAX/MJX code from LLMs can be brittle in practice. AURA compiles only after static validation against typed schemas governing the generated workflow, reward, randomization, and training files. Validation enforces: (i) type conformance (`int`, `float`, `bool`, `vector`, etc.); (ii) structural schema compliance; (iii) reference integrity (all state variables and sensors referenced in Φ_k exist in the MJX environment); and (iv) mathematical well-formedness of reward expressions. Expressions are built from a function registry (e.g., `fn.NORM_L2` producing JAX arrays) and must compile symbolically under MJX. A generation retry is triggered whenever schema validation detects an error. The schema validation generates a descriptive error message that includes the type of error, content of what caused the error, and sometimes a recommendation for how the error can be fixed. This error message, along with the previously generated (but invalid) YAML files, are fed back into the next attempt to guide corrections and avoid repeating mistakes; we set a maximum of five retries per stage. Once validated, YAMLs are compiled into MJX code.

F. Staged RL Training Loop

Compiled stages are trained with PPO across parallel simulation environments, each running for the episode limit specified in Θ_k. Policies from one stage initialize the next, enabling progressive skill learning.

G. Feedback and Iteration

Following each stage, an automated feedback module analyzes rollouts and training signals under a task-specific rubric (e.g., success rate, stability and energy, reward-component attribution). Its recommendations may adjust reward terms, randomizations, or hyperparameters for *subsequent stages within the same iteration* if the curriculum training has not yet completed all K stages.

An *iteration* of our framework is defined as the complete execution of: (i) VDB-backed retrieval and high-level planning; (ii) YAML generation and static validation for all stages; (iii) PPO training across K stages; with (iv) automated feedback analysis between each stage. After the final trained policy is deployed in simulation or on hardware, the user will evaluate the policy either based on the simulation video rollout or hardware performance, then provide *user feedback*: 2–4 sentences in natural language judging the policy's behavior and quality based on the video rollout (eg. "the policy survives well, but could track linear velocity better"). User feedback is attached to the files of the just-trained curriculum and inserted into the VDB alongside the YAMLs, metrics, and rollouts. Any training that incorporates user feedback begins a *new* iteration $(t{+}1)$, which again starts from VDB retrieval conditioned on the now-augmented database. Human input therefore does not mutate an ongoing run; it shapes the next planning phase via retrieval.

H. Modular LLM Collaboration

Hallucination mitigation: AURA limits LLM hallucinations through three mechanisms: (i) *retrieval grounding*—the planner is context-seeded by a single example selected from the top-3 VDB results; (ii) *schema constraints*—the model emits only typed YAML using a restricted operator registry (e.g., `fn.NORM_L2`) rather than arbitrary code; and (iii) *consistency checks*—static validation enforces type/structure and reference integrity to environment signals, followed by MJX compile checks and a capped regenerate-on-error loop (max 5 retries). Together, these mechanisms reject unsupported symbols and incoherent references, improve first-try success, and reduce off-distribution generations.

IV. Experiments

A. Simulation Environments

We present simulation experiments on a variety of tasks on three humanoid robots and a robot arm manipulator, all example AURA trained policy rollouts can be seen in Fig. 1. All AURA experiments are trained in MuJoCo-MJX and are evaluated over 1024 parallel environments on 5 seeds. CurricuLLM simulation results are reported based on their results in Isaac Lab.

1) Custom Humanoid: The custom humanoid locomotion environment exposes the velocity command, joint positions, projected gravity vector, last action, phase, and max foot height as observations and actuator gains, friction coefficients, body mass, COM position, foot contact geometry, initialization configuration, and roughness as domain randomization parameters. The simulation setup was adapted

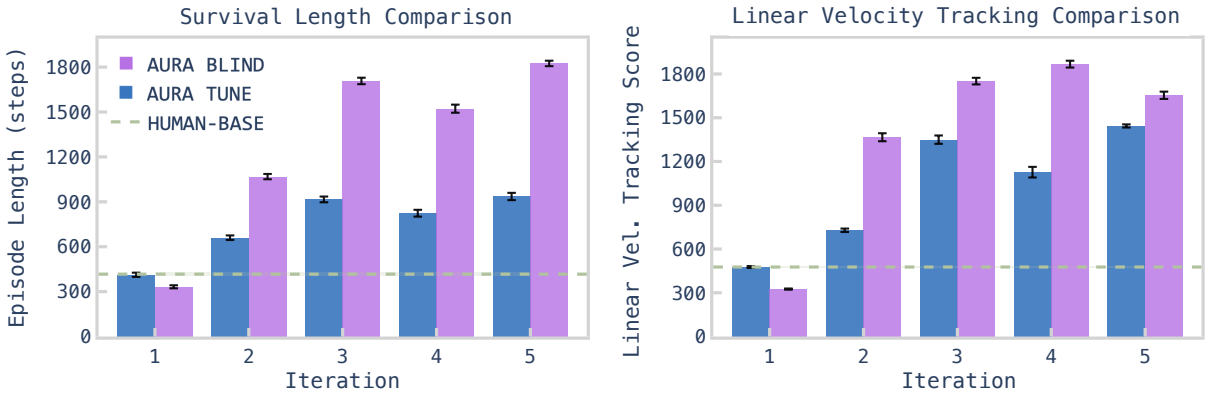

Fig. 3: Survival and linear velocity tracking scores across iterations to evaluate locomotion policy quality on a custom humanoid. The plots show the policy quality improvements of AURA over five iterations compared to MuJoCo Playground's expert designed Berkeley Humanoid reward and CuricuLLM's reported results in Isaac Lab. AURA Blind generates rewards from scratch (VDB is initialized as empty) and AURA Tune modifies and improves an existing reward designed for another embodiment (VDB is initialized with MuJoCo Playground's Berkeley Humanoid expert human rewards, domain randomizations, and training configuration).

	Episode Survival Length	Linear Vel. Tracking	Pushing Cube Success %
AURA	2873 ± 474.8	**2077** ± 425.6	**91.95** ± 0.95
Playground	**2903** ± 411.1	1546 ± 375.8	0.00 ± 0.00**
CurricuLLM*	315.4 ± 73.95	40.46 ± 10.64	62.50 ± 16.60

TABLE I: Policy evaluation across metrics. Episode survival length and linear velocity tracking are used to evaluate a velocity command following task on the Berkeley Humanoid. The success rate of pushing cubes is evaluated on the UR5e enviorment. *CurricuLLM's Berkeley Humanoid and Fetch-Push results are reported in the paper [24]. **MuJoCo Playground's Pushing Cube Success is reported using Franka Emika Panda rewards on the UR5e embodiment, which shouldn't be expected to be successful. AURA adapts the Franka expert reward and training configuration into an effective curriculum for the UR5e.

from the model in [54]. AURA is run for five iterations, with each iteration generating a new curriculum. Custom humanoid locomotion is evaluated with AURA Tune, which starts with the Berkeley Humanoid reward and adapts it into a curriculum that works on the custom humanoid hardware, and AURA Blind, which generates a curriculum from scratch. All environments are run at 50Hz, and the curriculum is limited to 400M training timesteps.

2) Berkeley Humanoid: For the Berkeley humanoid task, AURA uses the MuJoCo Playground's Berkeley Humanoid environment. Berkeley Humanoid's MuJoCo Playground policy is trained as detailed in [55]. The environment is run at 50Hz and both policies train over 300M steps.

3) BoosterT1 Humanoid: For BoosterT1 locomotion, AURA is trained and evaluated in the MuJoCo Playground BoosterT1 simulation environment, is run at 50Hz, and trains for 300M steps. BoosterT1's MuJoCo Playground policy is trained as detailed in [55].

4) UR5e Manipulator: The environment observations and domain randomizations are consistent with MuJoCo Playground's manipulator environment for the Franka Emika Panda (but with a UR5e robot). The environment frequency is 30Hz, and the curriculum is limited to 300M training steps. This environment is used for two tasks. In the pushing cube task the robot must push a block to a target position both of which are randomized in a 30cm by 30cm region (following CurricuLLM experiments). To be considered a success the center of the cube must be within 3cm of the target point. The stacking cube task begins with a cube in the gripper and the robot must place the cube on top of another block which is randomized in a 30cm by 30cm region.

B. Experimental Setup

We evaluate two aspects of AURA and baselines:

(1) Generation Success Rate. In Table II, we report the success rates of AURA, its ablations, CurricuLLM, and Eureka. For AURA, we measure *curriculum*-level success. A run is considered successful only if *all* stages (rewards, domain randomizations, and training configs) in the generated curriculum successfully launch GPU-accelerated training. AURA performs a single launch per task without parallel sampling. We also compare AURA with its ablated variants: AURA without a VDB, AURA without schema validation (no retires), and AURA using a single LLM for both high-level planning and YAML generation. Eureka and CurricuLLM launch multiple environments in parallel. We report the fraction of successful environment launches out of all generated rewards. Unlike AURA, success is measured one stage at a time rather than requiring a full curriculum to succeed.

(2) Training and Deployment Benchmarking. To evaluate policy quality, we run five iterations of curriculum generation and training using AURA, AURA Blind, AURA Tune, CurricuLLM, and a human-designed baseline. Each trained policy is evaluated with 5 random seeds each across 1024 randomized environments. Evaluation reward functions, domain perturbations, and task variations are hidden from all methods to simulate realistic zero-shot deployment. For locomotion, survival episode length is the average number of steps survived without the humanoid falling (max 3000).

	AURA	AURA w/o schema	AURA w/o VDB	AURA single-agent	CurricuLLM*	Eureka**
Training-launch Success Rate	**99 %**	47 %	38 %	7 %	31 %	12 / 49 %

TABLE II: Training-launch-success-rate comparing AURA and its ablated variants. All evaluations are conducted with GPT-4.1, as the original models used in the baselines are deprecated at the time of assessment. *CurricuLLM is evaluated on generating rewards for Berkeley Humanoid locomotion. **Eureka's 12% is evaluated on generation for their ANYmal task, which is most similar in complexity to humanoid robot tasks. Eureka's generation success rate across all available embodiments in their examples is 49% with simpler tasks generating more successfully.

Linear velocity tracking score is calculated per episode as:

$$\mathcal{S}_{\text{lin}} = \sum_{t=1}^{T} \exp\Big(-\frac{\|\mathbf{v}_t^{\text{cmd}} - \mathbf{v}_t^{\text{loc}}\|_2^2}{2(\sigma)^2}\Big),\ \sigma = 0.1$$

Where v_t^{cmd} and v_t^{loc} are the commanded and actual robot velocities at timestamp t and T is the number of timsteamps the policy survives for.

Selected final policies are transferred to hardware for qualitative testing and demonstration.

C. Simulation Experiment Evaluations

1) Custom Humanoid: Fig. 3 summarizes evaluation during simulation deployment across iterations, showing average survival and linear velocity tracking scores. Both AURA variants improve noticeably over the human baseline on survival and linear-velocity tracking, which shows that AURA can generate an effective curriculum from scratch and can adapt existing rewards into effective curriculums. AURA Tune and AURA Blind both iteratively improve over five iterations, reasoning about reward signals and user feedback. AURA Blind's curriculum did not generate or use any rewards related to foot swing or foot phase tracking, leading to a more inconsistent and asymmetrical gait, but better simulation performance compared to AURA Tune. The locomotion task prompt does not specify stylistic information about the gait, so this is a reasonable result. AURA Tune, which is initialized with MuJoCo Playground's rewards, kept existing reward terms for phase tracking, maintaining a consistent gait while still improving over the base reward.

2) Berkeley Humanoid: For the Berkeley Humanoid results shown in Table I, AURA's episode survival length is similar to the MuJoCo Playground baseline, showing that both survive nearly the full 3000-step horizon. The key difference is in linear velocity tracking, where AURA achieves a significantly better score, reflecting more precise command following. While Playground training schedules 200M steps of flat terrain then 100M steps on rough terrain, AURA split training into 100M on flat terrain with mild perturbations, 100M on rough terrain with mild perturbations, and 100M on rough terrain with heavier, more frequent pushes, while progressively expanding the command range up to the 1.0 m/s and 1.0 rad/s caps. AURA also made the velocity following reward more strict and increased reward emphasis on feet-phase and action-rate, resulting in better velocity command following and smoother, more consistent stepping.

3) BoosterT1 Humanoid: Training with the MuJoCo Playground's BoosterT1 reward directly resulted in a policy with jerky motion and small, high-frequency shakes during locomotion. AURA was told this as feedback, which led it to generate new reward terms which penalize the squared differences across consecutive time steps for actions and for velocities, resulting in a visibly smoother policy. Heavier domain randomization also helped improve performance, raising episode length from 2139 to 2366 steps and linear-velocity tracking score from 1786 to 2162.

4) UR5e Manipulator: For the UR5e cube pushing task, AURA begins with MuJoCo Playground's Franka Emika Panda environment reward which is not sufficient to train a policy in the UR5e environment as shown in Table I. AURA adapted the reward into a curriculum where the first stage has no positional domain randomization and reshaped the gripper–to-cube reward term to provide a denser reward signal over larger distances, encouraging progress toward the cube rather than only near contact. In the second stage, AURA reintroduced the positional domain randomization and increased the weight of the box to target reward. This led to a final policy which could consistently push the cube to the target location with over a 90% success rate.

For the cube stacking task AURA adapted the push task curriculum, dramatically decreasing the gripper-to-cube reward because the cube starts in the gripper, and emphasizing the cube-to-cube reward term. AURA achieved a mean task success rate of 72.7%.

D. Real-World Experiments

Hardware Setup: We use a custom 0.6 m-tall agile humanoid with 5-DoF legs and 3-DoF arms [56]. RL policies run at 50 Hz with sensors and actuators at 500 Hz. The end-to-end policy uses no estimators or low-level control beyond servo internal impedance controllers.

Fig. 1 shows a policy generated by AURA deployed on real hardware in a zero-shot setting. The outdoor locomotion policy shown in the figure is trained by AURA Tune, while the robot traversing a step is trained by AURA Blind. In response to a user prompt requesting outdoor-capable locomotion, AURA automatically generated a four-stage curriculum progressing from flat terrain with low randomization to rough terrain with high randomization, diverse velocity commands, and significant external perturbations. The resulting policy demonstrated robust zero-shot generalization when deployed outdoors, successfully handling uneven terrain and maintaining consistent gait tracking. The robot achieved walking at 0.18 m/s and average gait frequency of 1.91 Hz. It withstood substantial perturbations, recovering from lateral pushes up to 0.38 m/s and from being physically pushed off

a 50 mm high platform without falling, which surpassed the performance of existing manually tuned controllers [56].

E. Generation Success and Efficiency

1) Curriculum Generation: Table II presents the success rates of training runs for AURA, its ablated variants, and baseline methods including CurricuLLM and Eureka. AURA achieves a 99% success rate, outperforming all other methods and demonstrating the benefit of schema validation and RAG which allow for consistent generation of curriculums with hundreds of parameters and sometimes over ten unique reward terms.

2) Computation Efficiency: AURA requires only a *single* training launch per stage, since schema validation and VDB-guided retrieval ensure that nearly all generated curricula are executable on the first training launch. In contrast, Eureka and CurricuLLM each generate multiple training launches per stage—16 and 5, respectively—and retain only the best-performing successful run. As a result, AURA initiates far fewer training runs than either CurricuLLM or Eureka, leading to substantially lower computational cost for multi-stage workflows while still achieving competitive performance.

3) Multi-Agent LLM Setup: AURA's use of a multi-agent LLM setup—including a high-level curriculum generation agent and dedicated per-stage agents for reward, domain randomization, and configuration—was critical for producing complete and valid curricula. As shown in Table II, attempting to generate all stages in a single step is highly inconsistent. By decomposing the task into high-level planning and individual stage generation AURA forms smaller, more achievable tasks with higher success rates. The delegation of responsibilities also allows for individual stages to be retried in event of generation failure, rather than all stages.

V. Limitations

While AURA automates reward design, domain randomization, and curriculum-driven policy training, a nontrivial amount of human effort is still required. Setting up the MuJoCo simulation environment, defining observation spaces, action spaces, and environment state variables, remains a manual process that can impact policy success. Similarly, deploying the final policy to real hardware requires careful integration and validation.

Human Evaluation: Although AURA's training pipeline generates curricula autonomously, it still relies on human feedback at the end of each iteration to provide qualitative assessment of the deployed policy. While naively applying Vision-Language Models often fails to accurately describe robot data [57, 58], recent frameworks demonstrate promising methods for automatically analyzing robot trajectories and delivering feedback [59, 60]. These could be used in future work to fully automate feedback between iterations.

Exploration and VDB Content Bias: While AURA leverages its VDB to improve training reliability, this dependence introduces potential bias. If the database lacks diversity or contains inaccurate human feedback, subsequent retrieval-augmented generations may reinforce suboptimal patterns. AURA's curriculum search process has the potential to converge prematurely on a narrow set of solutions. Future work could incorporate explicit exploratory sampling of curricula designs or adversarial selection of prior examples to counteract this VDB content bias.

VI. Conclusion

This paper introduces **AURA (Autonomous Upskilling with Retrieval-Augmented Agents)**, a schema-centric curriculum RL framework leveraging multi-agent LLM architectures and RAG. By abstracting complex GPU-accelerated training pipelines into schema-validated YAML workflows, AURA reliably automates the design and refinement of multi-stage curricula, enabling policy training with iterative improvement. Empirical evaluations demonstrate that AURA outperforms baseline methods, achieving a higher generation success rate and superior policy performance across locomotion and manipulation tasks, and effective zero-shot hardware deployment. Real-world validation on a custom humanoid robot highlights AURA's capability to autonomously produce highly robust policies directly from user-defined prompts. The resulting controller demonstrated stable outdoor locomotion, gait tracking, and disturbance rejection, including recovery from lateral perturbations and vertical drops. Overall, AURA advances the development of scalable, generalizable prompt-to-policy frameworks, demonstrating the ability to substantially reduce the manual effort required in reward design while generating complex, high-performing curricula.

References

[1] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *Journal of Machine Learning Research*, vol. 21, no. 181, pp. 1–50, 2020.

[2] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement learning for versatile, dynamic, and robust bipedal locomotion control," *The International Journal of Robotics Research*, vol. 44, no. 5, pp. 840–888, 2025.

[3] D. Hoeller, N. Rudin, D. Sako, and M. Hutter, "Anymal parkour: Learning agile navigation for quadrupedal robots," *Science Robotics*, vol. 9, no. 88, eadi7566, 2024. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.adi7566.

[4] C. Li, E. Stanger-Jones, S. Heim, and S. Kim, *Fld: Fourier latent dynamics for structured motion representation and learning*, 2024. arXiv: 2402.13820 [cs.LG].

[5] S. Narvekar and P. Stone, "Learning curriculum policies for reinforcement learning," *arXiv preprint arXiv:1812.00285*, 2018.

[6] T. Haarnoja *et al.*, "Learning agile soccer skills for a bipedal robot with deep reinforcement learning," *Science Robotics*, vol. 9, no. 89, eadi8022, 2024.

[7] I. Radosavovic, T. Xiao, B. Zhang, T. Darrell, J. Malik, and K. Sreenath, "Real-world humanoid locomotion with reinforcement learning," *Science Robotics*, vol. 9, no. 89, eadi9579, 2024.

[8] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," *CoRR*, vol. abs/2109.11978, 2021. arXiv: 2109.11978.

[9] S. Chen, B. Zhang, M. W. Mueller, A. Rai, and K. Sreenath, *Learning torque control for quadrupedal locomotion*, 2023. arXiv: 2203.05194 [cs.RO].

[10] S. Jang and M. Han, "Combining reward shaping and curriculum learning for training agents with high dimensional continuous action spaces," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, 2018, pp. 1391–1393.

[11] S. M. Nguyen, N. Duminy, A. Manoury, D. Duhaut, and C. Buche, "Robots learn increasingly complex tasks with intrinsic motivation and automatic curriculum learning: Domain knowledge by emergence of affordances, hierarchical reinforcement and active imitation learning," *KI-Künstliche Intelligenz*, vol. 35, pp. 81–90, 2021.

[12] M. Li, J. Zhang, and E. Bareinboim, "Causally aligned curriculum learning," *arXiv preprint arXiv:2503.16799*, 2025.

[13] E. Sayar, G. Iacca, O. S. Oguz, and A. Knoll, "Diffusion-based curriculum reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 97 587–97 617, 2024.

[14] P. Soviany, R. T. Ionescu, P. Rota, and N. Sebe, "Curriculum learning: A survey," *International Journal of Computer Vision*, vol. 130, no. 6, pp. 1526–1565, 2022.

[15] Y. Hu *et al.*, "Learning to utilize shaping rewards: A new approach of reward shaping," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 931–15 941, 2020.

[16] A. Brohan *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv preprint arXiv:2212.06817*, 2022.

[17] M. Ahn *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[18] J. Liang *et al.*, "Code as policies: Language model programs for embodied control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 9493–9500.

[19] J. Sun, Q. Zhang, Y. Duan, X. Jiang, C. Cheng, and R. Xu, "Prompt, plan, perform: Llm-based humanoid control via quantized imitation learning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024, pp. 16 236–16 242.

[20] C. E. Mower *et al.*, "Ros-llm: A ros framework for embodied ai with task feedback and structured reasoning," *arXiv preprint arXiv:2406.19741*, 2024.

[21] R. Mon-Williams, G. Li, R. Long, W. Du, and C. G. Lucas, "Embodied large language models enable robots to complete complex tasks in unpredictable environments," *Nature Machine Intelligence*, pp. 1–10, 2025.

[22] G. R. Team *et al.*, "Gemini robotics: Bringing ai into the physical world," *arXiv preprint arXiv:2503.20020*, 2025.

[23] Y. J. Ma *et al.*, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv:2310.12931*, 2023.

[24] K. Ryu, Q. Liao, Z. Li, P. Delgosha, K. Sreenath, and N. Mehr, "Curricullm: Automatic task curricula design for learning complex robot skills using large language models," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2025, pp. 4470–4477.

[25] Y. J. Ma *et al.*, "Dreureka: Language model guided sim-to-real transfer," in *Robotics: Science and Systems (RSS)*, 2024.

[26] O. Erak, O. Alhussein, S. Naser, N. Alabbasi, D. Mi, and S. Muhaidat, "Large language model-driven curriculum design for mobile networks," in *2024 IEEE/CIC International Conference on Communications in China (ICCC)*, IEEE, Aug. 2024, pp. 179–184.

[27] Q. Liao, B. Zhang, X. Huang, X. Huang, Z. Li, and K. Sreenath, *Berkeley humanoid: A research platform for learning-based control*, arXiv preprint arXiv:2407.21781, urlhttps://arxiv.org/abs/2407.21781, 2024.

[28] B. Zitkovich *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*, PMLR, 2023, pp. 2165–2183.

[29] K. Black *et al.*, π_0*: A vision-language-action flow model for general robot control*, 2024. arXiv: 2410.24164 [cs.LG].

[30] O. M. Team *et al.*, *Octo: An open-source generalist robot policy*, 2024. arXiv: 2405.12213 [cs.RO].

[31] H. Huang *et al.*, *Otter: A vision-language-action model with text-aware visual feature extraction*, 2025. arXiv: 2503.03734 [cs.RO].

[32] J. Wang *et al.*, "Large language models for robotics: Opportunities, challenges, and perspectives," *Journal of Automation and Intelligence*, 2024.

[33] L. Cao, "Ai robots and humanoid ai: Review, perspectives and directions," *arXiv preprint arXiv:2405.15775*, 2024.

[34] R. Wang, Z. Yang, Z. Zhao, X. Tong, Z. Hong, and K. Qian, "Llm-based robot task planning with exceptional handling for general purpose service robots," in *2024 43rd Chinese Control Conference (CCC)*, IEEE, 2024, pp. 4439–4444.

[35] S. Rho, L. Smith, T. Li, S. Levine, X. B. Peng, and S. Ha, "Language guided skill discovery," *arXiv preprint arXiv:2406.06615*, 2024.

[36] J. Auerbach *et al.*, "Robogen: Robot generation through artificial evolution," in *Artificial Life Conference Proceedings*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ..., 2014, pp. 136–137.

[37] L. Diaz-Bone, M. Bagatella, J. Hübotter, and A. Krause, "Discover: Automated curricula for sparse-reward reinforcement learning," *arXiv preprint arXiv:2505.19850*, 2025.

[38] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *Conference on robot learning*, PMLR, 2017, pp. 482–495.

[39] S. Tao, A. Shukla, T.-k. Chan, and H. Su, "Reverse forward curriculum learning for extreme sample and demonstration efficiency in reinforcement learning," *arXiv preprint arXiv:2405.03379*, 2024.

[40] Z. Zhuang, S. Yao, and H. Zhao, *Humanoid parkour learning*, 2024. arXiv: 2406.10759 [cs.RO].

[41] Y. Ji *et al.*, "Hierarchical reinforcement learning for precise soccer shooting skills using a quadrupedal robot," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1479–1486, 2022.

[42] Y. Ji, G. B. Margolis, and P. Agrawal, "Dribblebot: Dynamic legged manipulation in the wild," *International Conference on Robotics and Automation*, 2023.

[43] D. Pavlichenko *et al.*, "Robocup 2022 adultsize winner nimbro: Upgraded perception, capture steps gait and phase-based in-walk kicks," in *Robot World Cup*, Springer, 2022, pp. 240–252.

[44] A. Adu-Bredu, G. Gibson, and J. Grizzle, "Exploring kinodynamic fabrics for reactive whole-body control of underactuated humanoid robots," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2023, pp. 10 397–10 404.

[45] W. Liang, S. Wang, H.-J. Wang, O. Bastani, D. Jayaraman, and Y. J. Ma, "Environment curriculum generation via large language models," in *Conference on Robot Learning (CoRL)*, 2024.

[46] P. Liu *et al.*, "Hm-rag: Hierarchical multi-agent multimodal retrieval augmented generation," *arXiv preprint arXiv:2504.12330*, 2025.

[47] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1.

[48] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[49] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2017, pp. 23–30.

[50] F. Muratore, C. Eilers, M. Gienger, and J. Peters, "Data-efficient domain randomization with bayesian optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 911–918, 2021.

[51] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5026–5033.

[52] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, *Brax - a differentiable physics engine for large scale rigid body simulation*, version 0.12.3, 2021.

[53] OpenAI, *New embedding models and api updates*, OpenAI Blog, Announcement of `text-embedding-3-small` and `text-embedding-3-large` models, Jan. 2024.

[54] Y. Tanaka, A. Zhu, Q. Wang, and D. Hong, "Mechanical intelligence-aware curriculum reinforcement learning for humanoids with parallel actuation," in *2025 IEEE-RAS 24th International Conference on Humanoid Robots (Humanoids)*, 2025, pp. 882–889.

[55] K. Zakka *et al.*, *Mujoco playground*, 2025. arXiv: 2502.08844 [cs.RO].

[56] Y. Liu, J. Shen, J. Zhang, X. Zhang, T. Zhu, and D. Hong, "Design and control of a miniature bipedal robot with proprioceptive actuation for dynamic behaviors," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 8547–8553.

[57] K. Chen, S. Xie, Z. Ma, P. R. Sanketi, and K. Goldberg, *Robo2vlm: Visual question answering from large-scale in-the-wild robot manipulation datasets*, 2025. arXiv: 2505.15517 [cs.RO].

[58] P. Sermanet *et al.*, *Robovqa: Multimodal long-horizon reasoning for robotics*, 2023. arXiv: 2311.00899 [cs.RO].

[59] Z. Zhou, P. Atreya, Y. L. Tan, K. Pertsch, and S. Levine, "Autoeval: Autonomous evaluation of generalist robot manipulation policies in the real world," *arXiv preprint arXiv:2503.24278*, 2025.

[60] J. Zhang *et al.*, "Rewind: Language-guided rewards teach robot policies without new demonstrations," *arXiv preprint arXiv:2505.10911*, 2025.