|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Title :** **Project Registration & Progress Review** | | **FF No. 180** |  |
|  | | | | | |
|  | | | | | |
| **Department:** Computer Engineering | | **Academic Year:** 2025-26 | | | |
| **Semester :** SEM-I | | **Group No. : SY F-15** | | | |
| **Project Title: EvoAI : Self-Evolving Generative AI Ecosystem** | | | | | |
| **Project Area: Artificial Intelligence, Multi-Agent Systems, Machine Learning** | | | | | |
| **Group Members Details:** | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sr. No. | Class & Div. | Roll No. | G.R.No. | Name of Student | Contact No. | Email ID |
| 1. | SY-F | 43 | 12413609 | Kaustubh Mukdam | 8849337271 | devidas.kaustubh24@vit.edu |
| 2. | SY-F | 62 | 12413603 | Om Kharate | 9172638800 | om.kharate241@vit.edu |
| 3. | SY-F | 63 | 12414597 | Ayush Khatal | 9604850960 | ayush.khatal24@vit.edu |
| 4. | SY-F | 66 | 12411770 | Samarth Khedkar | 8080663892 | samarth.khedkar24@vit.edu |

|  |
| --- |
| Name of Internal Guide: Vidya.S.Gaikwad Contact No. 9359889604  Email ID: vidya.gaikwad@vit.edu |

|  |
| --- |
| Project approved / Not approved  **Guide Project Coordinator Head of Department** |

T

**FF No** **180**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Project Synopsis: -** |  |  |  |  |  |
| **Synopsis: EvoAI: A Constitutional, ArchiveDriven SelfImproving Code Agent**  **1. Problem Statement**  The emergence of self-improving AI systems represents one of the most significant challenges and opportunities in artificial intelligence research. Large-scale self-improving agents such as Sakana AI's Darwin Gödel Machine deliver rapid capability gains but depend on expensive cloud clusters and frontier-size language models. Educational or small-lab settings cannot reproduce these results, leaving a critical research gap in **low-resource open-ended AI**. Moreover, unconstrained code-writing agents risk unsafe or incoherent self-modifications, so **alignment and controllability mechanisms**—notably Constitutional AI—must be embedded from day one.  EvoAI addresses two intertwined problems:   * **Safety Gap** — How can an evolving code agent be guided to stay helpful and harmless without costly human oversight, using constitutional principles and lightweight reinforcement learning with AI feedback (RLAIF)? * **Methodological Gap** — How can we develop systematic approaches to evaluate self-improving systems that balance computational efficiency with scientific rigor?   The problem is further complicated by the need to balance exploration and exploitation in the self-improvement process, ensure goal preservation across recursive modifications, and maintain interpretability as systems become more sophisticated through self-modification.  **2. Objectives**   1. **Design a minimal Darwin Gödel loop** that runs end-to-end on local hardware and supports empirical self-improvement with measurable capability gains. 2. **Embed Constitutional AI** to supply self-critique and revision rules, ensuring safe code edits, transparent decision logs, and alignment preservation throughout the self-improvement process. 3. **Integrate RLAIF-style selection** by rewarding patches that both pass tests and satisfy constitutional scores, without requiring full RLHF training infrastructure. 4. **Adopt a staged evaluation pipeline** of 10 → 40 → 100 SWE-bench tasks, mirroring Sakana's methodology but scaled to semester constraints and educational budgets. 5. **Quantitatively demonstrate improvement** from baseline (≈15%) to ≥45% success on the 100-task set within 6 months   **]**  **3. Novelty and Research Gap**  Based on our comprehensive analysis of recent literature, EvoAI addresses several critical gaps in the current landscape of self-improving AI systems:  **Research Gap Analysis**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Existing Method | Mathematical & Computational Basis | Usage Scenario | Strengths | Weaknesses | | **Gödel Machine** (Schmidhuber, 2006) | Formal proof systems; provably optimal self-modifications; bias-optimal proof search | Theoretical framework; unlimited computational resources | Mathematically rigorous; global optimality guarantees; | Computationally intractable; cannot act when utility unprovable; no practical implementation | | **Self-Taught Optimizer (STOP)** (Zelikman et al., 2024) | Language model scaffolding; recursive Python code editing; empirical validation | Small synthetic algorithmic tasks; frozen foundation models | First practical LM-based self-modifier; diverse search heuristics | Limited to narrow tasks; foundation model frozen; safety bypasses observed | | **Gödel Agent** (Yin et al., 2025) | Runtime memory inspection; monkey patching; recursive self-improvement routine | Reading comprehension, math, reasoning benchmarks | Practical self-referential implementation; continuous gains without meta-agent | Benchmark-bound validation; frozen LLM limits; specification gaming vulnerable | | **Darwin Gödel Machine** (Zhang et al., 2025) | Open-ended archive evolution; population-based search; empirical validation loops | Coding benchmarks; cloud infrastructure; frontier models | Strong empirical results; systematic evaluation; archive-based exploration | Prohibitive cost (~$22k/run); high resource requirements; limited to coding domain |     **Existing Methods Limitations**  Current approaches rely on either:   * **(i) Theoretical rigor but practical intractability** (Original Gödel Machine): Perfect mathematical foundations but cannot be implemented with realistic computational constraints * **(ii) Narrow practical implementations** (STOP, Gödel Agent): Work on limited domains with frozen foundation models, restricting self-improvement ceiling * **(iii) Resource-intensive systems** (Darwin Gödel Machine): Achieve strong results but require infrastructure beyond most research institutions   **Proposed Solution: EvoAI's Novel Contributions**  **EvoAI bridges theory and practice** by combining the *archive-based open-ended exploration* of the Darwin Gödel Machine with *constitutional safety mechanisms* and *resource-efficient evaluation strategies*. Our key innovations include:   * **Constitutional AI built-in**: a 10-rule engineering constitution governs every code edit, with mandatory critique→revision and a numeric compliance score used in selection. * **Compute-efficient staged evaluation**: 10→40→100 tasks with ≥40% and top-2 rating, d. * Archive-driven evolution: parent selection balances performance and diversity; lineage tracking and diversity metrics prevent premature convergence. * Accessible, modular framework: open-source code, subsets, and reproducible protocols with swappable modules (agent, archive, evaluator, constitution) for classrooms and small labs. * Methodology upgrades: transfer checks across models/languages, live safety/performance dashboard with audit logs, and standardized ablations (CAI/RLAIFlite on–off) for fair comparison.   **4. Methodology**  Our methodology is structured in six phases over 24 weeks, with each phase building on previous achievements while maintaining safety and reproducibility standards.  Phase 1: Literature & System Design (Weeks 1–4)   * Survey: Gödel Machine, STOP, Gödel Agent, Darwin Gödel Machine; Constitutional AI; RLAIF. * Engineering Constitution (10 rules): safe edits, reproducibility, minimal diffs, repo conventions, robust error handling/rollback, test mandates, time/memory limits, and logging. * Architecture: Modular, local-first deployment with CAI oversight and staged evaluation; clearly defined interfaces. * Risk Analysis: Sandbox constraints, likely failure modes, and mitigations.   Phase 2: Framework & Baseline (Weeks 5–8)   * Infra: Deploy 7–8B Llama via Ollama on RTX 4060 (4-bit quantization). * Agent (MVP): sandboxed execution, advanced file editing, git versioning, core tools/workflows. * Baselines: 10-task sanity + 40-task subset; log failures and performance. * Validation: Verify stability, resource profiles, and safety paths under load.   Phase 3: Constitutional AI + SelfModification (Weeks 9–12)   * CAI: Principleguided critique→revision before deployment; numeric compliance scoring. * SelfMod Pipeline: LLMguided code diffs under constitutional constraints; automated testing; rollback-on-fail. * Safety: Pentests of sandboxing, failure handling, and CAI guardrails.   Phase 4: RLAIFLite + Archive (Weeks 13–16)   * Archive Evolution: genealogy tracking; parent selection balances performance and diversity; promotion to full eval if ≥40% on 40-task subset and top2. * Staged Eval: Implement 10→40→100 with statistical monitoring to manage noise.   Phase 5: Integration, Optimization, Robustness (Weeks 17–20)   * EndtoEnd: Wire all components; CI-style runs. * Performance: Optimize RAM/GPU use and evaluation throughput. * Robustness: 25iteration endurance; adversarial probes; constitutional compliance under stress. * Transfer Prep: Set up crossmodel/domain test harness.   Phase 6: Final Evaluation, Ablations, Release (Weeks 21–24)   * Evaluation: 100-task SWEbench; 30-task Polyglot; transfer across FMs; significance testing. * Ablations: CAI on/off; RLAIFlite on/off; archive vs singleagent; reward variants. * Deliverables: Technical docs, tutorials, research paper, opensource repo with reproducible scripts.   Training Pipeline   1. Initialize zeroshot 7B model. 2. Constitutional selfcritique/revision exemplars. 3. Multisample generation (K=5) with CAI scoring. 4. Train lightweight preference model from heuristic rewards. 5. Archive evolution (select→modify→evaluate→update).   Model Design (Condensed)   * FM: Llama 3.1 8B Instruct, 4bit QLoRA; 4K–8K context. * Memory: Dynamic allocation; archiveaware scaling. * CAI: Embedded critiquerevision in generation loop. * Safety: Multilayer sandboxing, strict resource limits, continuous monitoring.   **Evaluation Metrics**  **Primary Metrics**:   * SWE-bench success rate (baseline → target ≥45%) * Polyglot cross-language performance * Constitutional compliance scores * Resource utilization efficiency   **5. Expected Outcomes**  **Quantitative Outcomes**   * **Performance Improvement**: Demonstrate ≥30% absolute improvement over baseline on 100-task SWE-bench subset (from ~15% to ≥45%) * **Cross-Language Generalization**: Show consistent improvements across Python, JavaScript, and other languages in Polyglot benchmark * **Constitutional Compliance**: Maintain >90% constitutional compliance rate across all self-modifications   **Scientific Contributions**   * **Methodological Advances**: Novel integration of Constitutional AI with recursive self-improvement * **Safety Research**: Practical approaches to alignment preservation in self-modifying systems * **Efficiency Research**: Strategies for effective self-improvement under computational constraints * **Evaluation Standards**: Standardized protocols for assessing self-improving AI systems   **6. Hardware and Software Specifications**  **Phase-by-Phase Hardware Requirements**   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Phase | CPU Cores | RAM (GB) | GPU VRAM | Primary Bottleneck | Rationale | | **Design (1-4)** | 4 | 8 | 0 | Human throughput | Literature review, system design, and specification writing | | **Baseline (5-8)** | 8 | 16 | 8 | GPU memory | Quantized 7B model deployment and basic inference testing | | **Self-Modification (9-12)** | 8 | 24 | 8 | System memory | Parallel diff generation, constitutional evaluation, and testing | | **RLAIF & Archive (13-16)** | 12 | 28 | 8 | CPU + memory | Multi-sample generation, preference scoring, and archive management | | **Integration (17-20)** | 12 | 32 | 8 | System memory | Full pipeline with archive history and parallel evaluation | | **Final Evaluation (21-24)** | 16 | 32 | 8-16 | GPU throughput | 100-task evaluation with optional cloud burst for time-sensitive experiments |     **Software Stack Specifications**  **Operating System**: Ubuntu 20.04 LTS or later (for Docker and development environment compatibility)  **Core Framework**:   * **Python**: 3.9+ with asyncio support for concurrent operations * **PyTorch**: 2.0+ with CUDA 11.8+ for GPU acceleration * **Transformers**: Latest Hugging Face library for model management * **Ollama**: For efficient local model serving and quantization   **Development Tools**:   * **Git**: Version control with LFS for model weights * **Docker**: Containerization for sandboxing and reproducibility * **Poetry/pip**: Dependency management and virtual environments * **Jupyter**: Interactive development and result visualization   **Monitoring and Logging**:   * **wandb/tensorboard**: Experiment tracking and visualization * **psutil**: System resource monitoring * **Custom logging**: Constitutional compliance and safety monitoring   **7. Limitations**  **Computational Complexity**  **Time Complexity**: Each self-improvement iteration requires O(K×T×M) operations where K=5 patches per problem, T=100 tasks in final evaluation, and M is the average model inference time. With current hardware, complete evaluation cycles require 1-2 hours per generation, potentially limiting the number of iterations possible within the semester timeframe.  **Space Complexity**: Archive growth is O(N×S) where N is the number of generated agents and S is the average storage per agent. With projected 50-100 agents over the semester, storage requirements will reach approximately 500GB for complete experiment logs and model checkpoints.  **Model Capacity and Reasoning Limitations**  **Foundation Model Constraints**: 7-8B parameter models have inherent limitations in complex reasoning and long-horizon planning compared to frontier models (GPT-4, Claude). This may result in a lower ceiling for achievable improvements and limit the sophistication of self-modifications.  **Evaluation and Validation Limitations**  **Benchmark Noise and Variance**: SWE-bench and Polyglot benchmarks exhibit inherent randomness due to LLM stochasticity. Our staged evaluation strategy (10→40→100 tasks) and 40% promotion threshold help mitigate this but cannot eliminate all noise-related false positives/negatives.  **Resource Constraints**  **Single-GPU Deployment**: Resource constraints prevent extensive parallel experimentation or large-scale parameter sweeps that might reveal optimal configurations.  **8. Conclusion**  Under resource and time constraints, EvoAI demonstrates that self-improving AI can be both practical and safe on consumer hardware by combining Gödelinspired selfmodification with Darwinstyle empirical evolution, all governed by Constitutional AI. The system operationalizes a full, repeatable improvement loop—parent selection, CAIguided critique→revision, multicandidate patching with RLAIFlite scoring, and staged evaluation (10→40→100 tasks)—to deliver measurable capability gains without frontierscale compute. Beyond raw benchmark improvements, EvoAI’s contribution is methodological and infrastructural: a transparent constitution for code agents, strict sandboxing and rollback for safety, an archive with genealogy and diversity to prevent premature convergence, and open, reproducible protocols that lower the barrier to entry for students and small labs.  This work advances alignment for selfmodifying systems by making safety an intrinsic part of the improvement loop rather than an external afterthought, and it provides reusable evaluation tools—staged gating to tame stochasticity, transfer analyses across models/languages, and efficiency optimizations tailored to 8GB VRAM. While acknowledging limits in compute, model capacity, and variance, EvoAI mitigates risk through constitutional audits, ablations (CAI/RLAIFlite on–off), comprehensive logging, and conservative promotion criteria. As a result, EvoAI is more than a semester prototype: it is a replicable foundation for future extensions (larger models, multiagent roles, broader domains), and a concrete step toward beneficial, aligned, and controllable selfimproving AI—judged not only by higher pass rates, but by its scientific rigor, educational accessibility, and safety-first design. | | | | |  |

**FF No** **180**

|  |  |  |  |
| --- | --- | --- | --- |
| Group No. |  | | |
| Activity | Review Schedule | Progress Review Report submitted | Signature of Guide |
| Review 1 | Mid Sem. Semester | Yes / No |  |
| Review 2 | End of Semester | Yes / No |  |

Format of Progress Review Report:

|  |
| --- |
| **Review No.: 1 Group No.: Date:** |
| **Progress Review Report** |
| **Signature of Guide:** |

|  |
| --- |
| **Review No.: 2 Group No.: Date:** |
| **Progress Review Report** |
| **Signature of Guide:** |