PROJECT PROPOSAL

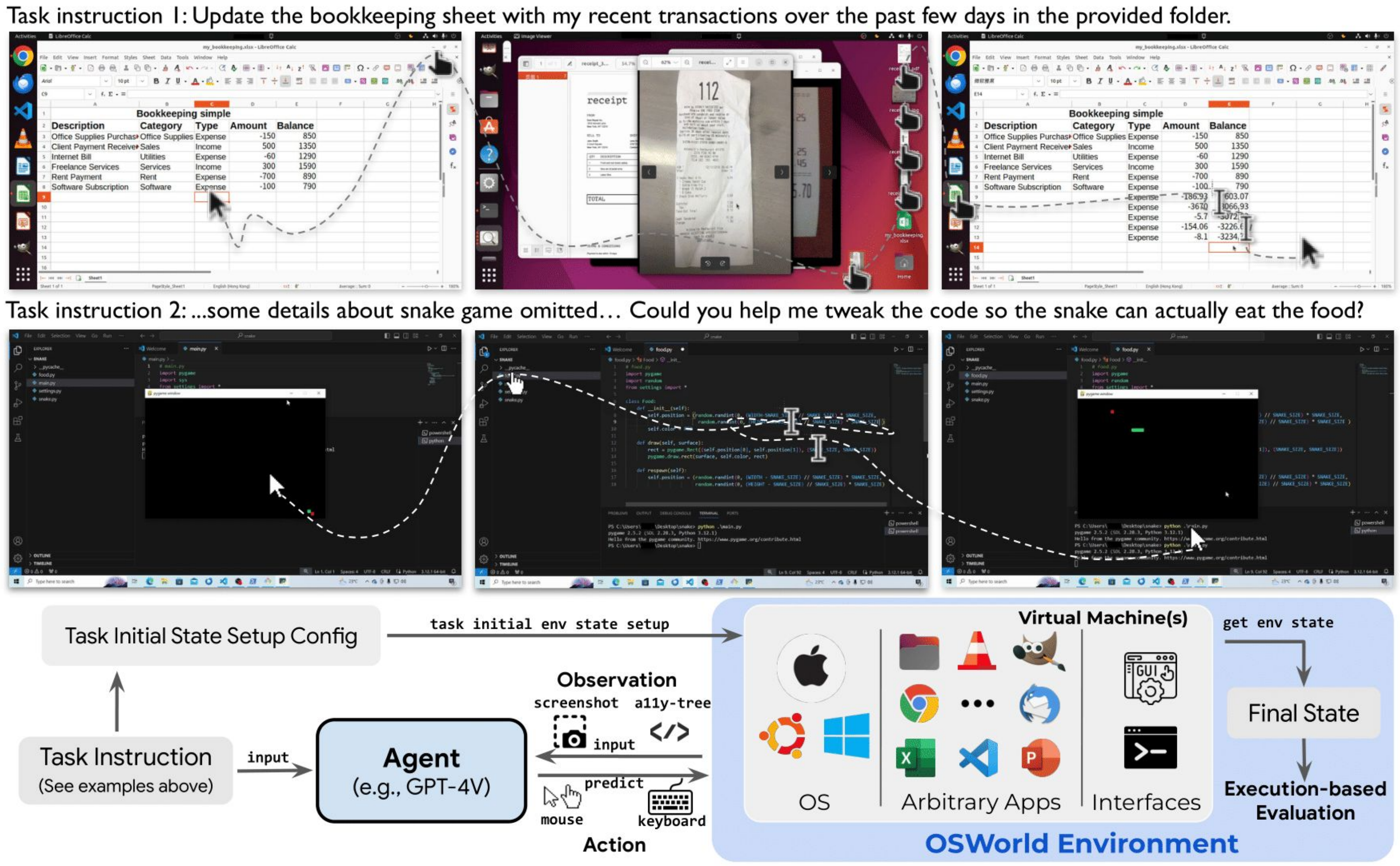
Idea -1

A-RISE

Action-centric, Reliable, Intelligent, Small-model Engine

An **open-source** computer-use agent for research automation, dataset creation, and note-taking.

Computer Use Agents[CUA]


Task instruction 1: Update the bookkeeping sheet with my recent transactions over the past few days in the provided folder.
Task instruction 2: ...some details about snake game omitted... Could you help me tweak the code so the snake can actually eat the food?
Task Initial State Setup Config
task initial env state setup
Virtual Machine(s)
get env state
Observation
screenshot
a11y-tree
input
Task Instruction
(See examples above)
input
Agent
(e.g., GPT-4V)
predict
mouse
keyboard
Action
OS
Arbitrary Apps
Interfaces
Final State
Execution-based Evaluation
OSWorld Environment

Bookkeeping simple

Description	Category	Type	Amount	Balance
Office Supplies Purchas	Office Supplies	Expense	-150	850
Client Payment Receive	Sales	Income	500	1350
Internet Bill	Utilities	Expense	-60	1290
Freelance Services	Services	Income	300	1590
Rent Payment	Rent	Expense	-700	890
Software Subscription	Software	Expense	-100	790
		Expense	-186.93	603.07
		Expense	-3670	-3066.93
		Expense	-5.7	-3072.[illegible]
		Expense	-154.06	-3226.[illegible]
		Expense	-8.1	-3234.[illegible]

```
# food.py
import pygame
import random
from settings import *

class Food:
    def __init__(self):
        self.position = (random.randint(0, (WIDTH-SNAKE_SIZE) // SNAKE_SIZE) * SNAKE_SIZE,
                         random.randint(0, (HEIGHT - SNAKE_SIZE) // SNAKE_SIZE) * SNAKE_SIZE)
        self.color = ...

    def draw(self, surface):
        rect = pygame.Rect((self.position[0], self.position[1]), (SNAKE_SIZE, SNAKE_SIZE))
        pygame.draw.rect(surface, self.color, rect)

    def respawn(self):
        self.position = (random.randint(0, (WIDTH - SNAKE_SIZE) // SNAKE_SIZE) * SNAKE_SIZE,
                         random.randint(0, (HEIGHT - SNAKE_SIZE) // SNAKE_SIZE) * SNAKE_SIZE)
```

```
PS C:\Users\[redacted]\Desktop\snake> python .\main.py
pygame 2.5.2 (SDL 2.28.3, Python 3.12.1)
Hello from the pygame community. https://www.pygame.org/contribute.html
PS C:\Users\[redacted]\Desktop\snake>
```

Fig-1: - Basic CUA Agent Example

State of the Art (SOTA) Techniques

Paper	Core Idea	Key Strengths	One-Line Example of How It Works
Agent S	Experience-augmented **hierarchical planning** with **memory + retrieval + ACI**	• Combines past experiences + web knowledge for planning • Uses **Agent-Computer Interface (ACI)** for precise, safe actions • Best at multi-step workflows	Breaks “delete email account” into subtasks → retrieves online + past strategies → clicks correct buttons via ACI safely.
ComputerRL	**Massive-scale RL** on parallel desktops + **API-GUI hybrid actions**	• Trains on **thousands of VMs** asynchronously • Uses both GUI clicks + app APIs intelligently • Achieves **48.1% OSWorld-Verified** with open models	Decides: “Should I call Thunderbird API or click?” → Picks API if exposed → falls back to GUI clicks if needed.
MobileAgent-v 3 (GUI-Owl)	Fully **end-to-end VLM policy** trained across devices	• Unifies perception → planning → action into a single model • Runs on Android, macOS, Ubuntu, Windows • Uses **self-evolving trajectory generation**	Looks at Thunderbird screenshot → directly predicts “Open Settings → Select Account → Remove Account” without explicit planner.
OpenCUA	**Reflective reasoning agent** trained on massive **AGENTNET dataset**	• Uses **reflective 3-level chain-of-thought** (perception → plan → action) • Open-source, strong benchmarks • AGENTNET tool collects cross-OS trajectories	Generates an internal “thought → plan → action” trace → executes clicks in Thunderbird → stores self-reflection for reuse.

Table-1(part-1): - Current SOTA, and their strengths

PC Agent-E	**Trajectory Boost** with **small human seed dataset** + synthetic augmentation	• Starts from **312 human demos**, then uses strong teacher models to **generate diverse variants** • Extremely **data-efficient** • Outperforms GPT-4o on WindowsAgentArena	Given 312 Thunderbird account-removal examples → asks Claude to generate alternate strategies → trains on combined trajectories.
SEAgent	**Self-evolving agent** with curriculum-based learning & multi-agent setup	• Uses **Actor + World State Model + Curriculum Generator** • Trains from easy to hard tasks automatically • Combines GRPO RL + imitation on failures	Actor explores Thunderbird UI → fails → World State Model captions why → Curriculum Generator breaks into easier subtasks → retries until success.
UI-TARS	Combines **System-2 reasoning**, **multi-platform action unification**, and large-scale trace learning	• Decomposes tasks into **milestones** • Collects **millions of online GUI traces** • Uses iterative **reflection + correction loops**	Predicts milestones like “Go to Settings → Select Account → Remove” → executes sequential actions with verification checkpoints.
PC Agent-E + WindowsArena-V2	Data-efficient agent optimized for **Windows GUI automation**	• Uses **ReAct scaffold**: Thought → Action → Observation • Executes actions with **PyAutoGUI** • Optimized specifically for **Windows UI apps**	Reads Thunderbird UI → predicts “Click Account Settings” → executes via PyAutoGUI → rechecks the updated UI state.

Table-1(part-2): - Current SOTA, and their strengths

What is the problem ?

Existing Limitations and Gaps

Modern **Computer-Use Agents (CUAs)** aim to control desktops, browsers, and apps like humans. However, **current systems face major limitations**

Current research agents (Agent S, UI-TARS, ComputerRL) are **powerful but fragmented** — they require **huge datasets, expensive models, and per-app customization**.

Open Gap	Why It Matters
1. Adaptive Planning	No system decides when to **search**, **think**, or **act** dynamically.
2. Better Memory Systems	Agents forget past context; no persistent **GUI-aware memory** to speed up repeated tasks.
3. Verified Reflection	Existing reflection is mostly **narrative**; there's no automatic **check and repair** when a plan fails.
4. On-Prem + Small Models	Most rely on **GPT/Claude-sized models**, making them hard to deploy locally or securely.
5. Efficient Data Usage	Current systems need **huge datasets**; there's a gap in **data-efficient training** strategies.
6. Generalization to New GUIs	Most models fail when the UI **changes slightly** or the **workflow differs**.
7. Unified Connector Layer	No common standard to interact with **apps, browsers, files, and APIs** — everything is custom-built.

Table-2: - Open Gaps in CUA agents

Paper	Current Limitations	Gaps / Opportunities
Agent S	• Fails on **long, multi-step tasks** if one subtask goes wrong • Relies too much on **web search** and old memory • No **adaptive planner**; tasks are executed in a fixed order • Errors in **grounding UI elements** due to OCR & accessibility tree mismatches • Actions are slow since it performs **one step at a time**	• Build an **adaptive planner** that can replan mid-task • Improve **visual + structural grounding** for UI elements • Add **reflection** to detect failures and repair plans automatically
ComputerRL	• Needs **huge compute** and thousands of virtual machines • Heavily depends on **reinforcement learning**, which is costly • Relies on APIs; fails when apps don't expose them • No **memory** of past user actions • No mechanism to **check and fix plans**	• Build a **lightweight system** that works on fewer resources • Combine **API + GUI** actions into **reusable skills** • Add **persistent memory** for faster task completion
MobileAgent-v3 (GUI-Owl)	• Works like a **black box** — no clear explanation of decisions • Lacks a separate **planner**; hard to control • Struggles when **UI layouts change** • Performs poorly when the task needs **external knowledge**	• Combine **visual models with planners** for better control • Add **knowledge retrieval** when external info is needed • Make models **adaptable** to unseen interfaces
OpenCUA	• Uses **reflective reasoning**, but it's **narrative**, not verified • Needs a **large curated dataset**; expensive to maintain • Struggles when UI layouts change • Still has **low success rates** on long workflows	• Build **state-based reflection** to verify if steps worked • Use **smaller datasets + synthetic data generation** to train agents • Create **memory systems** that handle UI changes dynamically

Table-3(part-1): - Current SOTA limitations and gaps

PC Agent-E	• Starts with a **small dataset** → depends heavily on **synthetic data** quality • Limited to **Windows apps** • No deep **reflection** to handle failures • Cannot handle unseen layouts well	• Improve **synthetic data quality** using better generation methods • Extend to **cross-platform** GUIs • Add **memory + plan repair** for better performance
SEAgent	• Uses a fragile **world model** to judge success; often inaccurate • Rewards are **noisy** for long tasks • May **overfit** to easy subtasks • Struggles on **real-world apps** with complex layouts	• Build better **multi-signal evaluators** (visual + text + functional) • Improve **reward shaping** for complex tasks • Make agents learn **general skills** that transfer across apps
UI-TARS	• Needs **millions of training examples** → very resource-heavy • System-2 reasoning fails on **pop-ups** & **dynamic UI changes** • Reflection lacks **proper verification** • Cannot decide **when to think, search, or act**	• Build **causal reflection** to verify outcomes • Add an **adaptive planner** that switches between reasoning, retrieval, and action • Create **data-efficient training methods**

Table-3(part-2): - Current SOTA limitations and gaps

CONCLUSION: Why Current Solutions Are Not Enough

1. Heavy Dependence on Large Cloud Models

- Most systems (Agent S, UI-TARS, ComputerRL) rely on **GPT-4o, Claude, or 32B+ VLMs**.
- High **inference cost**, **high token usage**, and limited **on-prem deployment**.
- **Example:** Agent S doubles success rate but uses GPT-4o + Claude → impractical for real-time tasks.

2. Latency & Inefficiency

- Current agents make **multiple sequential LLM calls per action**.
- **Agent S** executes **one GUI action per step** → long workflows are painfully slow.
- **UI-TARS** and **MobileAgent-v3** fail to optimize scheduling → unsuitable for live automation.

3. Weak Grounding & UI Adaptability

- Heavy reliance on **accessibility trees + OCR** (Agent S, OpenCUA).
- No dynamic switching between **DOM trees, screenshots, and cached UI bindings**.
- **ComputerRL** fails when APIs are missing; **MobileAgent-v3** struggles on unseen GUIs.

4. Session-Bound Agents

- No persistent, cross-session memory in most systems.
- Every workflow starts **from scratch** → repeated grounding, repeated planning.
- **Agent S** has episodic memory but lacks persistent, GUI-aware memory; **OpenCUA** suffers the same.

5. Poor Safety & Verification

- Reflection today is **narrative** (OpenCUA, UI-TARS) — agents “assume” success instead of verifying it.
- **No causal state-diff checks** to confirm actions succeeded.
- **No rollback mechanisms** for failed subtasks → cascading errors in multi-step tasks.

6. Evaluation Gaps

- Most papers optimize **success rate only** → ignoring:
  - **Latency** (time to complete task)
  - **Cost** (LLM calls per workflow)
  - **Energy usage** (GPU cycles on repeated inference)
- Example: Agent S succeeds in more tasks but requires **2–3× longer wall-clock time** than needed.

7. Poor Data & Sample Efficiency

- **UI-TARS** and **OpenCUA** need **millions of curated demos** to reach modest success rates.
- **PC Agent-E** shows potential for synthetic augmentation but lacks quality filtering.
- No system integrates **data-efficient training** + **plan reuse** effectively.

8. Lack of Standardization

- Every paper builds **custom connectors** for apps, browsers, and APIs.
- No **common middleware layer** → hard to extend, hard to collaborate.
- Leads to redundant engineering across research efforts.

Our Idea/Solution

Our solution focuses on **systems-level improvements** that make agents **scalable, lightweight, and reproducible**.

A **unified, open-source platform** for **computer-use agents** that can **plan, act, and learn** across any desktop or web application by combining:

- **Adaptive Planning** → Dynamically decide when to **think**, **search**, or **act**
- **Persistent GUI Memory** → Remembers UI layouts & past actions across sessions
- **Causal Reflection** → Verifies results and **repairs plans automatically**
- **Universal Connector Layer** → Single interface to control **apps, browsers, files, and APIs**
- **Optimized for On-Prem** → Runs efficiently on **small open-source multimodal models,** Using **small open models** (quantized 2B–7B)
- Prioritizing **efficiency, reliability, and safety** at the **systems** level.

Simply put

An **open-source**, **action-centric**, and **on-device-first** runtime that:

- Minimizes **model calls** via **cheap-first action routing**.
- Learns & **reuses plans across sessions** using **persistent memory**.
- Handles **UI changes robustly** with **delta-aware partial replanning**.
- Adds built-in **safety, verification, and rollback**.
- Evaluates efficiency using **steps, tokens, latency, and energy**.

Applications: - Automating research, dataset creation, and note-taking workflows.

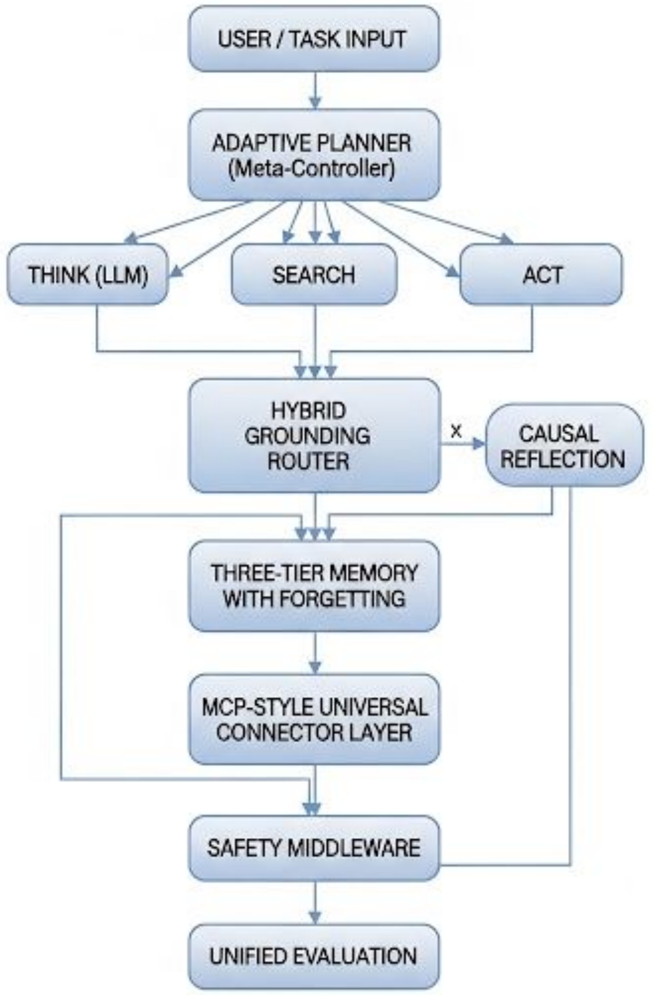


Fig-2: - Basic flow of Solution

Systems Component (Novelty)

Novel Component	What It Is	System-Level Contribution	Why It’s Novel vs Existing Papers
1. MCP-Based Connector Layer	A universal middleware that standardizes app, browser, API, and OS actions.	Cross-app interoperability, faster integration, and unified automation APIs.	Agent S, UI-TARS, and OpenCUA use **custom per-app connectors**; A-RISE introduces the **first standardized schema**.
2. Persistent GUI Memory Graph	A GUI-aware memory graph storing selectors, embeddings, and OCR spans.	Enables **stateful, cross-session caching**; speeds up recurring workflows and UI grounding.	No persistent GUI-level memory in existing works; Agent S/OpenCUA keep only task-bound episodic traces.
3. Adaptive Planner + Execution Cache	A meta-controller routes between **retrieval, local reasoning, and GUI actions** dynamically.	Reduces redundant model calls, improves **efficiency**, optimizes workflow routing in real time.	Existing systems use **static DAGs** (Agent S) or rely fully on **monolithic VLMs**; no dynamic orchestration exists.
4. Causal Reflection Engine	Uses **state-diff verification** (DOM + screenshots) to confirm success and trigger rollbacks.	Adds **verifiable correctness**, **self-healing plans**, and safe recovery without LLM retries.	OpenCUA/UI-TARS use **narrative reflection** only; A-RISE is the **first to verify and repair causally**.

Table-3(part-1): - Systems Contribution

5. Workflow Optimizer (Graph Scheduling)	Models workflows as a graph; optimizes execution, enables **parallel subtasks** where possible.	Decreases wall-clock time, reduces model + connector calls, and improves concurrency.	Agent S executes steps **sequentially**; no paper does **graph-based workflow scheduling**.
6. Cross-Session Plan & Retrieval Cache	Stores **plans, retrievals, and grounding hits**; reuses them across sessions.	Improves speed and cost-efficiency for **recurring or similar tasks**.	No existing work caches **plans**; A-RISE introduces inference reuse across users and sessions.
7. Small On-Prem Multi-Modal Models	Fine-tuned **7B–13B multimodal models** for GUI grounding + planning; optimized with ONNX/TensorRT.	Enables **offline, secure deployment**, reduces token cost, and speeds up inference.	Existing papers depend on **GPT-4o/Claude**; A-RISE is the **first efficient open-source on-device stack**.
8. Safety Middleware + Policy Gating	Risk-aware executor layer with **pre-action verification, rollback snapshots, and confirm prompts**.	Prevents accidental destructive operations and ensures task reliability.	None of the current systems integrate **policy-driven safety** as a first-class system component.
9. Unified Evaluation Harness	Logs **Success@k, Steps, Tokens, Latency, Energy**; includes **delta-resilience** and safety tests.	Provides **publishable systems metrics** showing efficiency improvements.	Existing benchmarks ignore cost, energy, and robustness; A-RISE is the **first efficiency-focused harness**.

Table-3(part-2): - Systems Contribution

Prioritization: Memory + Caching

Persistent GUI Memory Graph

- Store DOM selectors, embeddings, OCR spans as a graph.
- Reuse past GUI traces → faster grounding, fewer LLM calls.

Cross-Session Plan Cache

- Save successful task plans (e.g., “remove account → settings > accounts > delete”).
- Retrieve and replay on future tasks → efficiency + reduced errors.

Forgetting & Efficiency Policies

- **TTL expiry**: drop outdated traces.
- **Usage-based decay**: reinforce reused, fade unused.
- **Failure invalidation**: downrank bad strategies.
- **Consolidation**: merge repetitive traces into higher-level summaries.

Prioritization II: Safety + Verification

Causal Reflection Engine

- Verify actions with **state-diff checks** (DOM + screenshots).
- If mismatch → rollback, retry, or safe stop.
- Guarantees correctness beyond LLM “self-reflection.”

Safety Middleware + Policy Gating

- Pre-action checks for risky operations (delete, submit, overwrite).
- Confirm or block unsafe actions.
- Combine with rollback snapshots → prevents destructive errors.

Ideas that Might be Deprioritized

- **MCP Connector Layer**
  - Would require rewriting connectors for many apps.
  - High engineering effort → better suited as long-term infra work.

- **Workflow Optimizer (Graph Scheduling)**
  - True parallelization across subtasks is complex.
  - Needs orchestration layer + thread-safety; unlikely in one semester.

- **Small On-Prem Multi-Modal Models**
  - Requires fine-tuning or optimization (ONNX/TensorRT).
  - Heavy infra, more about efficiency engineering than research novelty.

Example Workflow

Task: “From Jira & Gmail, compile September bug triage:

1. Pull all ‘P1 open’ tickets this month
2. Export CSV
3. Cross-link with Gmail customer emails
4. Summarize results into Google Docs
5. Email final doc to team”

Step 1 — Jira Dashboard

- UI hash matched → **cache hit** → instantly applies saved filters.
- New popup breaks flow → **state-diff fails**, DOM ranker dismisses it → **no model call needed**.

Step 2 — Export CSV

- Recipe “Jira CSV export” found in **plan cache** → reuses path → **0 tokens used**.
- File downloaded; verification predicate passes.

Step 3 — Gmail Lookup

- Gmail layout changed → **cache miss** → tiny VLM grounds new FAB button in **1 step**, DOM text confirms → resume.

Step 4 — Summarize in Docs

- Docs UI changed → FAB replaced “New Doc” → **partial re-plan** switches to keyboard shortcut → **no full restart**.
- Small LLM generates summary template; inserts matched rows.

Step 5 — Email Output

- Two-man rule prompts user before sending team mail → passes → **workflow complete**.

Impact:

- **LLM calls:** 2 (plan + summary).
- **VLM calls:** 1 (Gmail FAB).
- **Cache hit rate:** 60%.
- **Latency reduced by ~43%**, **tokens reduced by ~61%**, **energy saved ~35%**.
- Handles dynamic layouts, cross-app workflows, and persistence across sessions.

How will we Evaluate our Solution ?

Existing Open Source Benchmarks

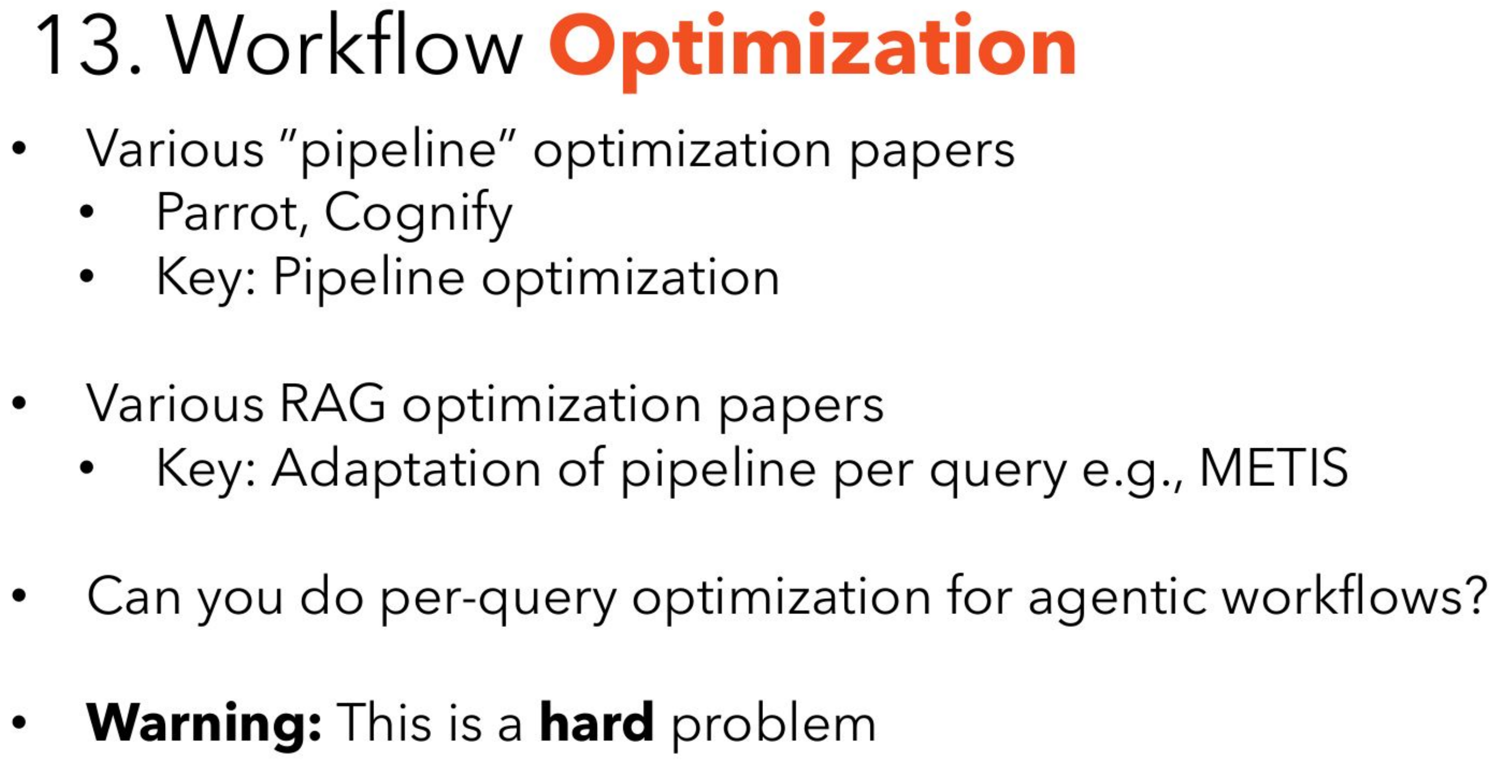
- **Benchmarks:**
  - **OSWorld** → 369 cross-OS tasks; big human vs agent gap.
  - **Windows Agent Arena** → 150+ Windows tasks, shows poor generalization.
  - **BrowserGym / WebArena** → Web-focused testing.
- **Popular Open-Source Agents:**
  - **Agent S** → Best OSS numbers (~56% OSWorld success) but server-heavy.
  - **Browser-Use** → Popular for scraping but limited reasoning.
  - **OpenCUA** → Comprehensive stack, but training + inference expensive.

How to Demonstrate Improvement
• Show fewer API failures & faster task completion vs Agent S • Demonstrate reduced development effort when adding new apps
• Measure latency reduction in GUI grounding • Show success-rate gains on repeated tasks vs baselines
• Compare wall-clock execution times on OSWorld tasks • Show fewer LLM calls & reduced cost per task
• Measure **Fix@k** (fraction of failures recovered) • Show success-rate improvements on long workflows
• Compare total completion time vs Agent S/UI-TARS • Visualize fewer environment round-trips per task
• Measure speedup & cost savings across repeated Thunderbird or LibreOffice tasks
• Show inference speedups on-device • Demonstrate competitive accuracy vs GPT-4-based Agent S

THANK YOU !

Idea -2 [OLD] [Scrapped]

What is the Problem ?



13. Workflow **Optimization**

- Various "pipeline" optimization papers
  - Parrot, Cognify
  - Key: Pipeline optimization

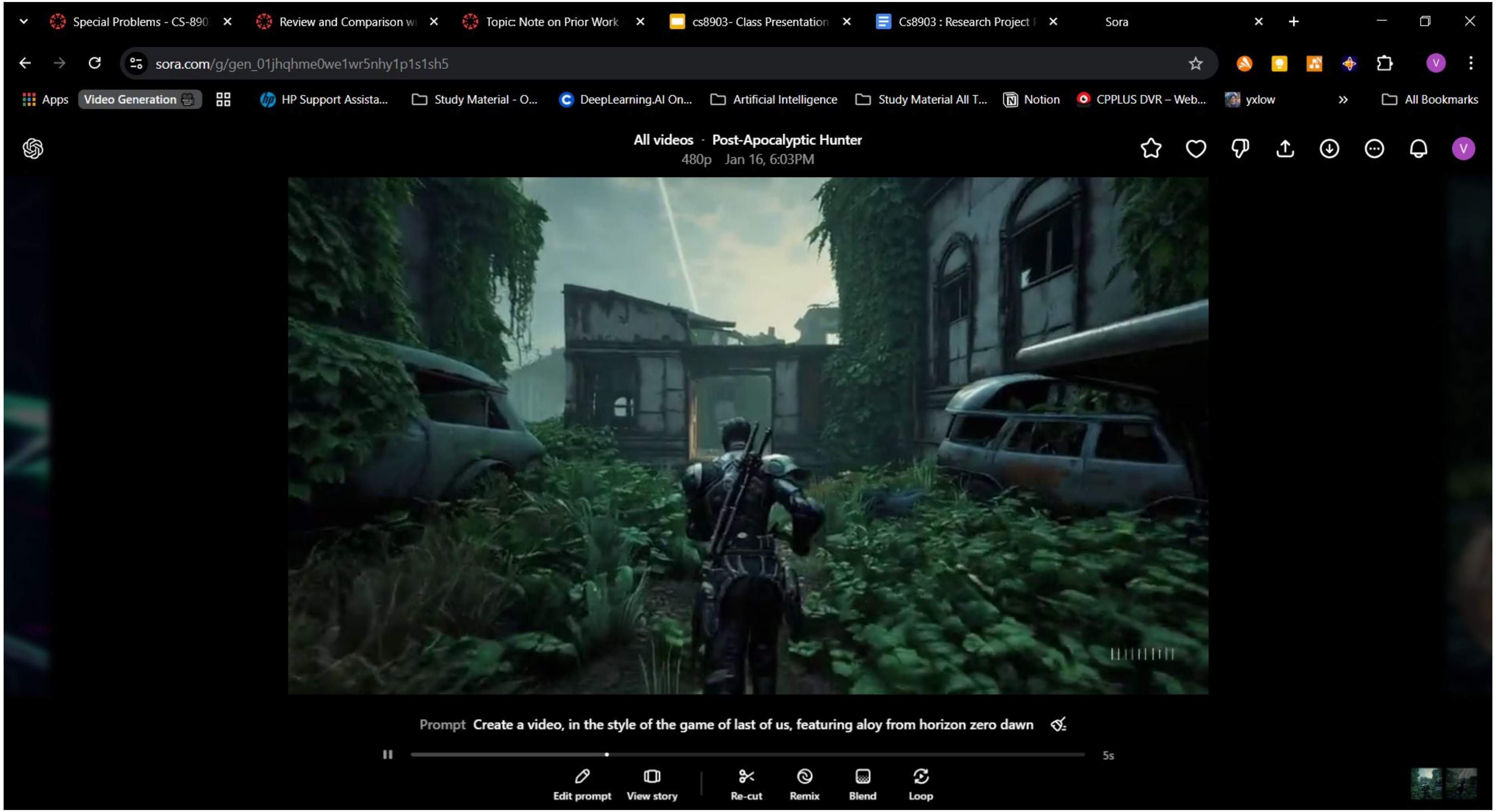
- Various RAG optimization papers
  - Key: Adaptation of pipeline per query e.g., METIS

- Can you do per-query optimization for agentic workflows?

- **Warning:** This is a **hard** problem

Current Problems in AI Video Generation

Model	Strengths	Limitations	Adaptability	Orchestration
OpenAI Sora	Photorealistic, long coherent videos (minutes); high fidelity	Closed-source, black-box pipeline	❌ Fixed	❌ Single-shot
Google Veo 3	Cinematic quality, good compositional control	Limited flexibility, not open	❌ Fixed	❌ Single-shot
Tencent Janus	Lightweight, fast inference, research-focused	Primarily single-shot, less multimodal control	❌ Fixed	❌ Single-shot



sora.com/g/gen_01jhqhme0we1wr5nhy1p1s1sh5

All videos · Post-Apocalyptic Hunter

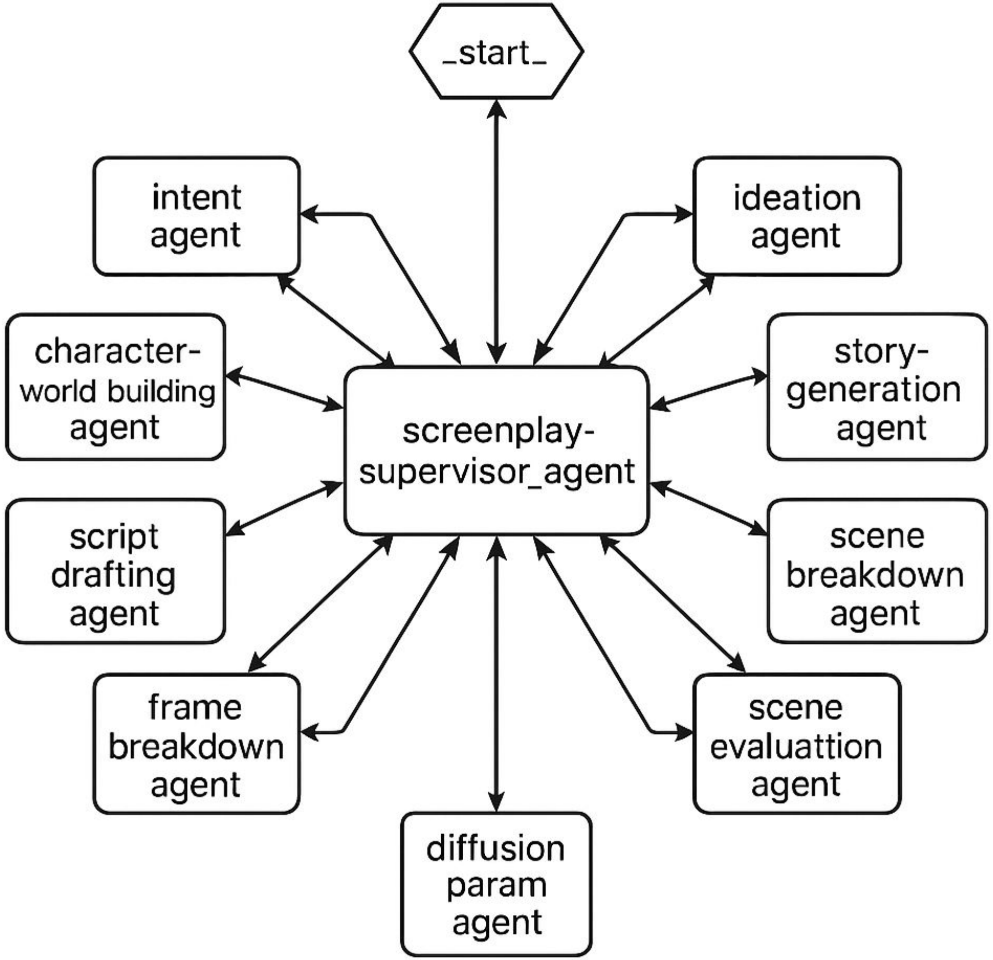
480p Jan 16, 6:03PM

Prompt Create a video, in the style of the game of last of us, featuring aloy from horizon zero dawn

Previous Background: ATHENA

What is ATHENA?
A **multi-agent generative AI system** for dynamic screenplay generation, multimodal synthesis, and memory-driven orchestration.
Combines **LLMs, diffusion models, memory agents, reflective supervisors**, and evaluation modules.

Core Features
Agent orchestration: modular tasks (screenplay → image generation → alignment).
Memory-driven reasoning: Redis-based vector + JSON stores for long/short-term memory.
Evaluation loops: CLIP/FID metrics to guide retries and quality assurance.



Existing ATHENA Limitations: -

Static Workflows

- Uses **pre-defined DAGs** → every query follows the same pipeline.
- Lacks ability to **re-plan** or choose alternate routes dynamically.

Memory Challenges

- Currently Relies on JSON(in context memory) only.
- No **hierarchical memory** (short-term vs long-term vs global).
- Risk of **staleness, inconsistency, and retrieval overhead**.

Planner Weakness

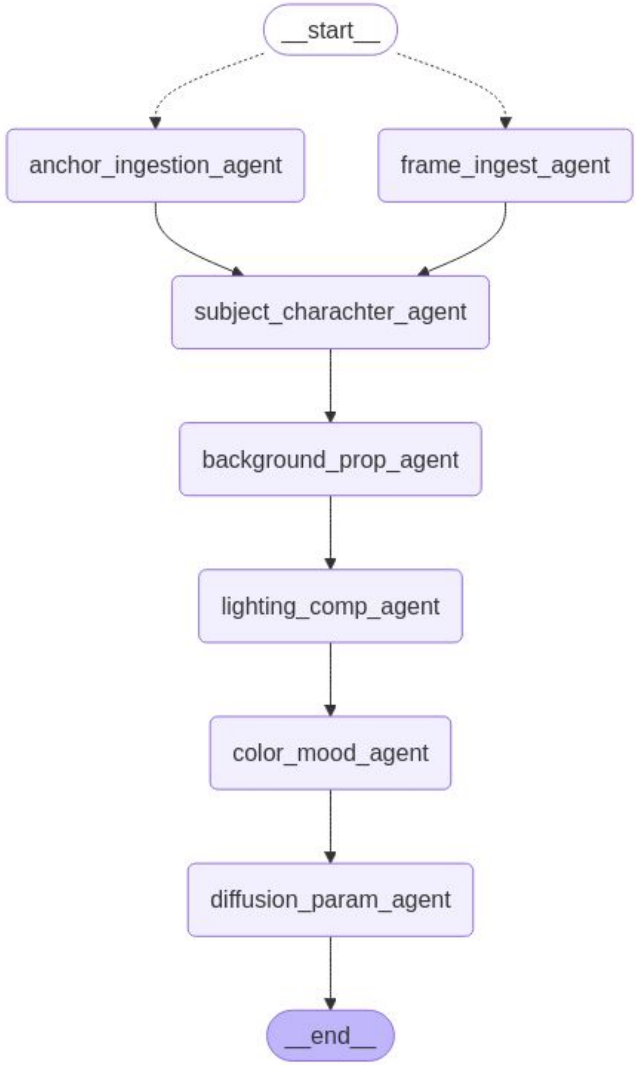
- Current planner is **manual** (fixed ordering of agents).
- No **per-query adaptive planning** or graph pruning.

Parallelism Gaps

- Independent agents (e.g., multiple image renders) are not fully parallelized.
- Results in **higher latency** and wasted compute.

System Fragility

- Failures in one node can cascade → limited **fault tolerance**.
- Retry logic is coarse, not cost-aware.



Solution Proposed:

Vision
Transform ATHENA into a **dynamic, per-query adaptive system** that generates and optimizes workflows at runtime.

Planned Enhancements

1. **Dynamic DAG Generation**
   - Build workflows *on the fly* based on query type, available tools, and model capabilities.
2. **Adaptive Planner & Executor**
   - Replace static ordering with a **planner that selects best subgraph** for each query.
3. **Parallel Execution**
   - Enable concurrent execution of independent agents (e.g., multiple image renders).
4. **Memory Hierarchy**
   - Introduce **short-term, long-term, and global memory layers** with consistency policies.
5. **Fault-Tolerant Scheduling**
   - Intelligent retries, fallbacks to smaller/cheaper models, and **re-planning on failures**.
6. **Cost & Quality Awareness**
   - Route tasks based on **compute budget, latency requirements, and quality thresholds**.

Expected Outcome :- Smarter, leaner, anc resilient workflows tailored **per query.**

Systems Component

- **Workflow Compiler**
  - Translates user request → candidate dynamic DAG.
  - Encodes dependencies, parallelizable tasks, and model/tool availability.
- **Execution Engine**
  - Schedules tasks across compute nodes.
  - Handles retries, backpressure, and adaptive subgraph execution.
- **Memory Layer**
  - Hierarchical design:
    - **Short-term (session cache)** for immediate context.
    - **Long-term (vector DB)** for historical knowledge.
    - **Global (knowledge graph/DB)** for persistent facts.
  - Provides **provenance + consistency** across agents.
- **Resource Manager**
  - Monitors GPU/CPU budgets and latency constraints.
  - Routes tasks to models based on **cost vs quality trade-offs**.
- **Observability & Metrics**
  - Tracks latency, success/failure rates, compute costs.
  - Provides **feedback loop** for optimization of future runs.