Flash-Searcher: Fast and Effective Web Agents via DAG-Based Parallel Execution

OPPO Agent Team

Motivation: Why Are Existing Agents Slow?

Typical Agent Loop

- Plan
- Execute
- Reflect
- Verify
- Re-execute

Key Bottlenecks

- Strong sequential dependency
- Redundant search and repeated verification
- Long execution chains for deep research tasks
- Better reliability often comes at the cost of higher latency

Question: Can we preserve strong task-solving ability while substantially reducing execution steps and end-to-end delay?

Key Observation: Complex Tasks Are Not Strictly Linear

Independent sub-goals

Independent sub-goals that can be pursued in parallel.

Weakly coupled verification paths

Weakly coupled verification paths with partial dependencies.

Auxiliary information retrieval

Auxiliary information that can be retrieved early.

Conclusion: A complex task should **not always** be executed as a simple chain.

Core Idea: Move from a sequential chain to a dependency-aware DAG

Sequential Chain

Strictly linear execution

Dependency-aware DAG

Parallel & Dependency-aware

Framework Overview

END-TO-END FLOW

Task → DAG-based Plan → Parallel Execution → Progress Summary → Final Answer

Core Components

DAG-based Plan Construction

Constructing execution plans with explicit dependencies and candidate paths.

Parallel Execution

Running ready subtasks in parallel with coordinated tool orchestration.

Adaptive Progress Tracking

Periodically summarizing progress and revising unresolved branches.

Design Goals

Maximize safe parallelism
Safety and usefulness prioritized

Preserve logical dependencies
Maintain logical consistency

Maintain execution state
Stability under long horizons

Module 1: DAG-Based Plan Construction

Input & Output

Input: composite task T
Output: DAG plan G = (V, E)

Node Types

Goal Nodes: Objectives the agent must complete
Path Nodes: Candidate solution strategies

Edge Semantics

Dependency and prerequisite relations.

Operational Plan

This is not just branching thought generation; it is a plan for real tool execution.

Module 2: Parallel Execution and Tool Orchestration

Dynamic Scheduling Strategy

- At each step, select ready subtasks from the pending set
- A subtask may be scheduled when:
- All prerequisites are complete
- Partial execution can still provide useful auxiliary signals

Parallel Execution

- Execute multiple subtasks in parallel

Minimal Tool Stack Design

Search Tool: Serper API

Crawl Tool: Jina Reader with automatic summarization

"The gain comes from better orchestration, not from simply adding a large number of tools."

Module 3:
Progress Tracking, Replanning, and Relaxed Constraints

Periodic Replanning

- Every Δ steps, the plan is updated
- Remove completed nodes
- Revalidate unresolved dependencies
- Insert new nodes if needed

Relaxed Constraints

- Retrieve auxiliary signals before full prerequisites are available
- Later validate them with consistency checks

Key Benefits

- Reduced idle time
- Controlled error propagation
- Better long-context management

Framework Summary

The framework maintains coherence through adaptive progress tracking and periodic replanning. It balances speed and correctness by allowing early auxiliary retrieval under relaxed constraints, followed by consistency checks to ensure accuracy.

Why the Method Works

Parallel Progress

Convert serial waiting into parallel progress, eliminating idle time.

Shared Evidence

Replace redundant search with shared evidence and cross-validation.

Adaptive Summary

Replace long unstructured context with periodic summaries and graph updates.

Core Argument: DAG planning + parallel tools + adaptive summarization improves both **quality** and **efficiency**.

Experimental Setup

Four Benchmarks

- **BrowseComp:** Large-scale hard web retrieval
- **xbench-DeepSearch:** Expert-written deep search
- **GAIA:** Complex general assistant tasks
- **HLE:** High-difficulty reasoning

Tool Configuration

- **Search:** Core retrieval capability
- **Crawl:** Web crawling with auto-summarization using the same backbone model

Evaluation Metrics

- **LLM-as-Judge:** Automated assessment
- **Judge Model:** GPT-4.1-mini
- **Main Metric:** Pass@1

The experiments are designed to test the framework's generalization across multiple demanding settings, not just narrow scenarios. We use a simple tool setup to isolate the effect of the framework itself.

Main Results: Performance

BrowseComp

67.7

Near OpenAI ChatGPT (68.9)

xbench-DS

83.0

Outperforms BrowseMaster & MetaSO

GAIA

82.5

Strong across diverse task types

HLE

44.0

State-of-the-art result

Key Takeaway: Consistency & Generalization

The framework demonstrates strong performance across retrieval-heavy tasks and harder reasoning-oriented settings. This suggests the DAG-based execution strategy is broadly useful for complex web-agent workloads, making the agent both faster and stronger in a general way.

Efficiency Results: Steps, Time, and Cost

Compared with OAgents and sequential ReAct, Flash-Searcher reduces overhead substantially.

Faster Execution

GAIA Steps:
11.2 → 7.4 (-35%)
BrowseComp Time:
27.4m → 9.6m

xbench-DeepSearch

Steps Reduction:
37.8 → **11.4**
Time Reduction:
12.9m → **4.9m**

Higher Utilization

Flash-Searcher:
3.00 calls/step
Baselines (OAgents/OWL):
~0.84 calls/step

Lower Cost per Query

xbench: $0.13 → **$0.07** | GAIA: $0.05 → **$0.03**

Key Insight

Flash-Searcher achieves efficiency not by doing less work, but by making each step more productive and dense.

A Second Contribution: Distilling Parallel Reasoning into Models

Beyond Framework Design

The paper also studies learning, aiming to distill parallel reasoning into models.

Curated Training Data

3,354 DAG-based trajectories from multiple web-agent sources.

Lightweight Fine-tuning

Transferring Flash-Searcher structure into model behavior.

Performance Results

Model / BrowseComp / xbench-DS / GAIA / HLE

Qwen-2.5-32B: **14.4 / 63.0 / 57.3 / 19.4**

Qwen-2.5-72B: **18.9 / 68.0 / 61.2 / 20.2**

Key Implication

Parallel reasoning can become a**learnable inductive bias**, not just a runtime trick.

"Our contribution is not limited to an execution framework. The graph-based parallel reasoning pattern itself may be a learnable inductive bias, which opens the door to stronger standalone search agents."

Limitations & Takeaways

Key Limitations

- Parallelization introduces overhead on simple tasks
- Smaller update interval improves accuracy but increases cost
- Code tool brings limited benefit in web-centered settings

Concrete Examples

- **Bamboogle:**Flash-Searcher matches ReAct score but is slower
- **HLE:**Code tool integration adds marginal gain with higher runtime

Final Takeaway

Shift focus from "How do we build stronger agents?" to**"How do we design better execution structure?"**

Conclusion

Flash-Searcher redefines the efficiency-effectiveness frontier of web agents through**DAG-based parallel execution**.

Thank you!