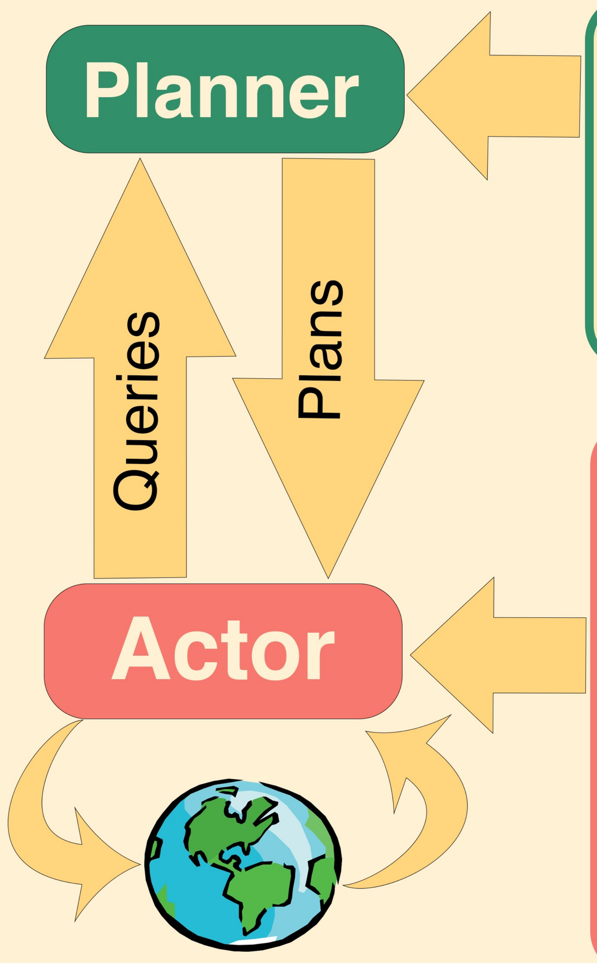
A Demo of Integrated Acting, Planning, and Learning with Hierarchical Operational Models

Sunandita Patra[1], James Mason[1], Malik Ghallab[2], Paolo Traverso[3], Dana Nau[1]

[1]University of Maryland, USA, [2]LAAS-CNRS, France, [3]FBK-ICT, Italy.

Planning
- Prediction + search
- To reach a goal or accomplish a task

Acting
- Performing tasks and actions in the real world
- Adapt to context, react to events
- Dynamic, partially observable environment
- Wrong move can lead to failures and dead ends
- Needs online help from planner

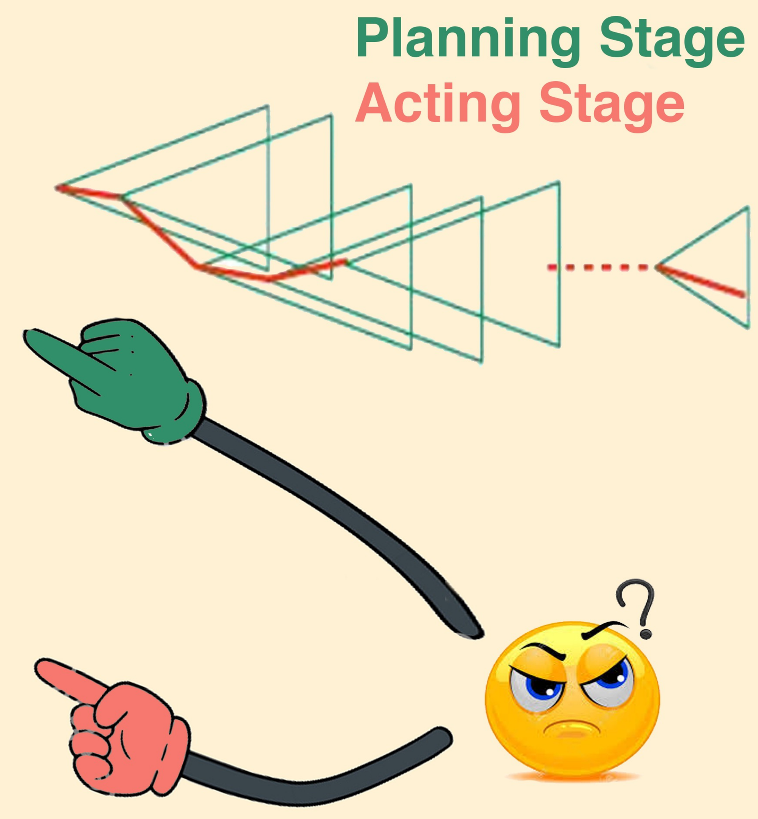


Descriptive Models
What the actions do?

eg. PDDL actions
action: action-identifier
pre: test
effects: $effect_1$, $effect_2$, ..., $effect_n$

Operational Models
How to perform the task on actor's execution platform?

method-name(arg_1, ..., arg_k)
task: task-identifier
pre: test
body: computer program to generate commands and more tasks



Problem: The two models may not be consistent
- Can't verify or manage plans
- Acting suffers

Acting Algorithm: RAE

RAE = **R**efinement **A**cting **E**ngine

loop:
- for every new task
- Candidates <- {applicable method instances}
- choose m from Candidates
- create a refinement stack
  - like a program execution stack
  - initially with just task and m
- add the stack to Agenda
- for each stack in Agenda
  - Progress(stack)

Use UPOM to make an informed choice

rescue(p)

Possible choices
...

rescue-method1(p, r)
task: rescue(p)
pre: status(r) = Free and loc(p) = Unknown
body: for l in LOCATIONS:
move(r, l)
sense(l)
if loc(p) = l:
help(r, p)
return
output("cannot find" p)

rescue-method2(p,l)
task: rescue(p)
pre: loc(p) = l and l ≠ Unknown
body: r <- free robot nearest to p
if r != NONE:
move(r, l)
help(r, p)
else fail

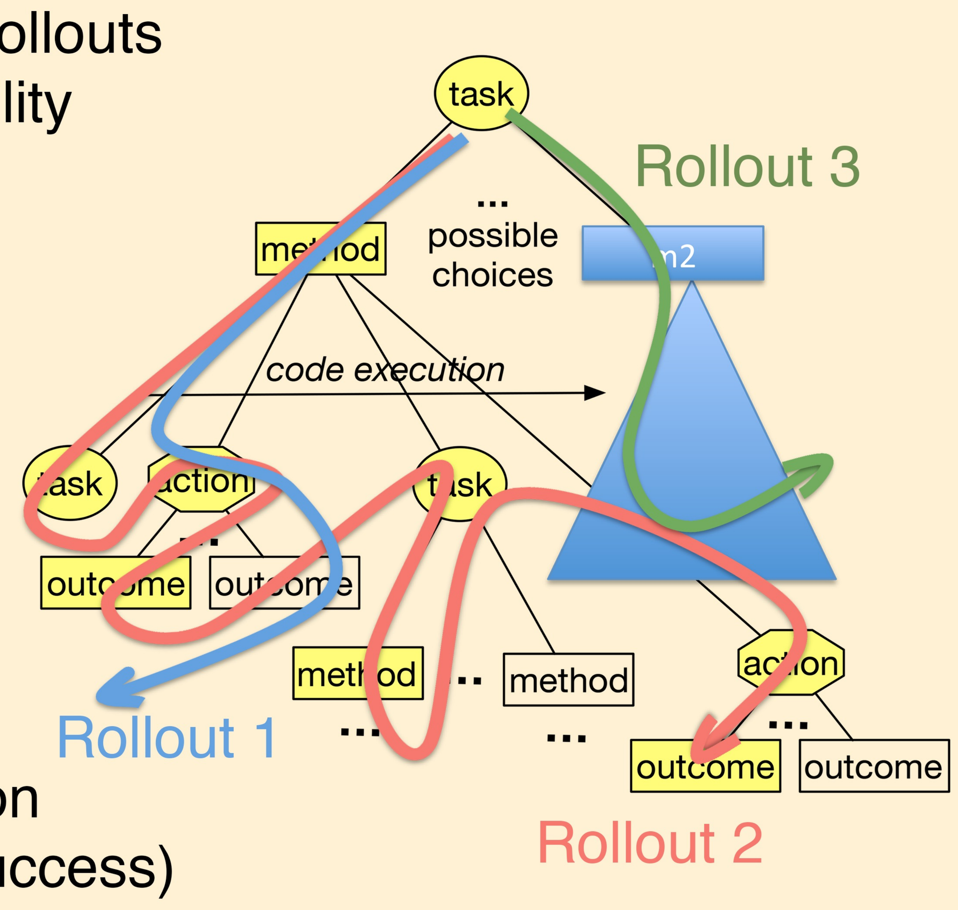
Our Contributions:
- Planner UPOM that uses the actor's operational models for planning
- Learning strategies integrated with actor and planner

Planning Procedure: UPOM

Idea: Execute the applicable refinement methods in a simulated environment
- Do several Monte Carlo rollouts
- Estimate the expected utility for every choice
- Choose the method with highest expected utility

UPOM handles one rollout
- A UCT-like procedure
- Balances exploration vs exploitation

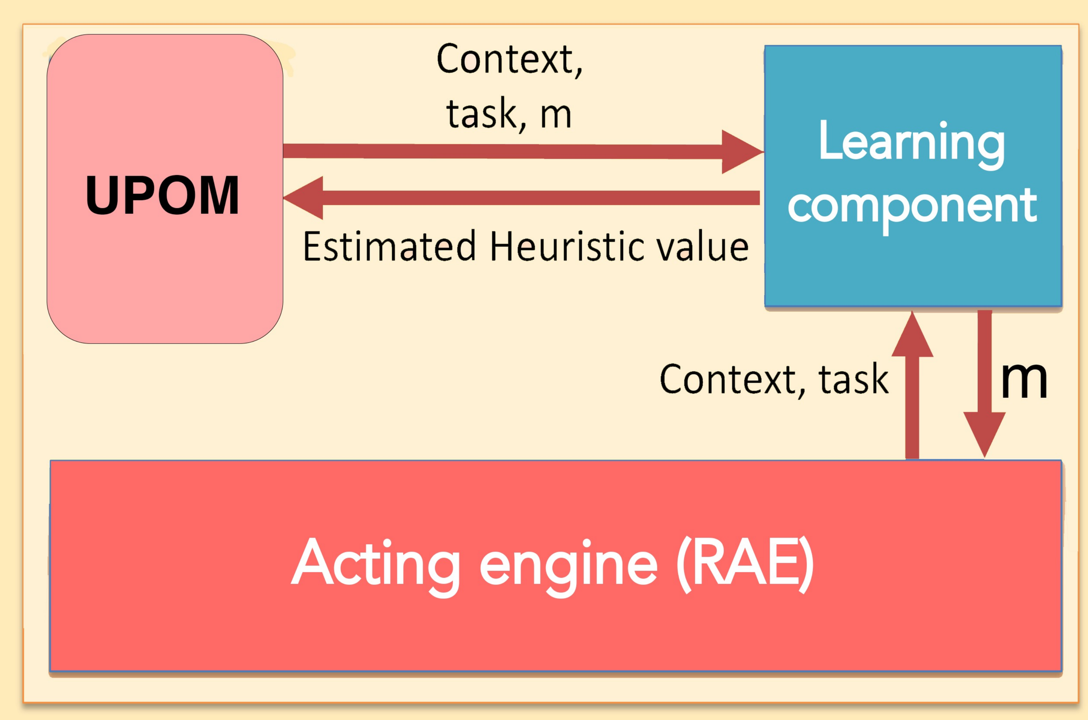
Utility: User-defined function (e.g., cost, probability of success)



Learning Strategies: Learnπ and LearnH

Learnπ:
To choose a refinement method for a task

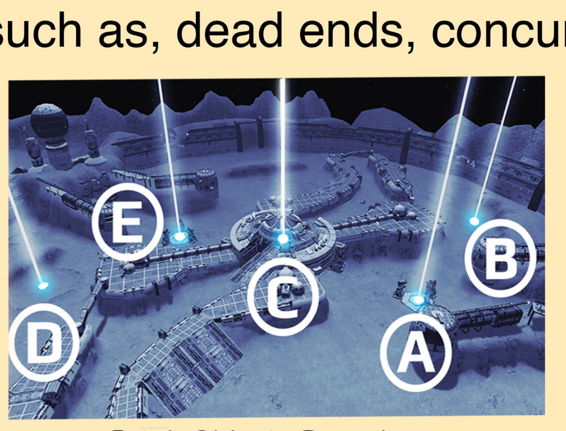
LearnH:
To estimate a heuristic for UPOM

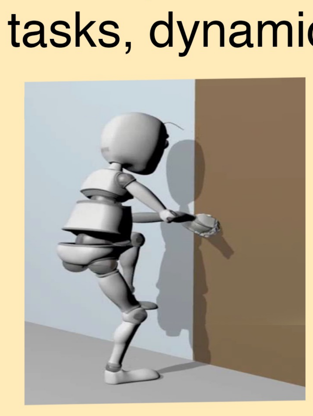


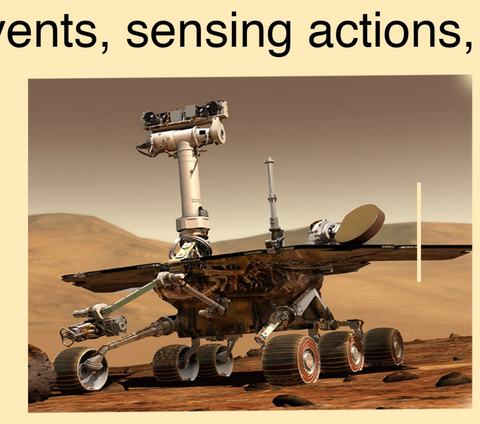
- Gather training data from acting and planning traces of RAE and UPOM
- Train classifiers (multi-layered perceptrons)

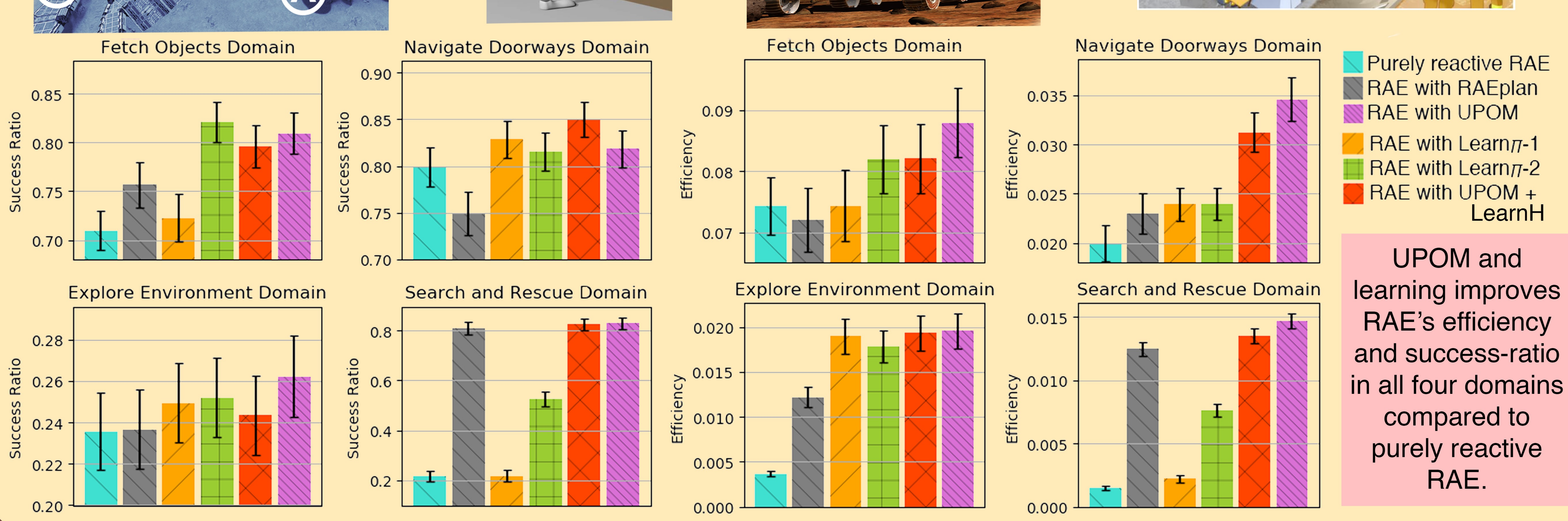
Experimental Evaluation

Measured efficiency (reciprocal of cost) and success ratio in four simulated domains with different properties, such as, dead ends, concurrent tasks, dynamic events, sensing actions, agent collaboration, dynamic events.









UPOM and learning improves RAE's efficiency and success-ratio in all four domains compared to purely reactive RAE.

Conclusions
- Using same model for both acting and planning is useful
- Key idea: Use operational models for planning instead of descriptive models
- Avoids inconsistency between actor and planner
- RAE with UPOM / Learnπ / LearnH shows improved performance compared to purely reactive RAE in four simulated domains

Contact:
Sunandita Patra
patras@umd.edu
University of Maryland, College Park
https://sunanditapatra.wixsite.com/camp