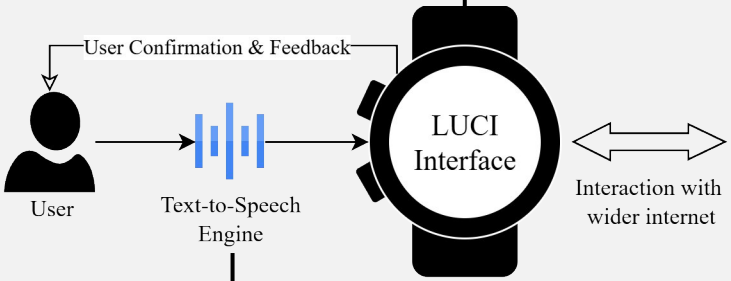
LUCI: Lightweight UI Command Interface

Guna Lagudu, Vinayak Sharma, **Aviral Shrivastava**
Arizona State University

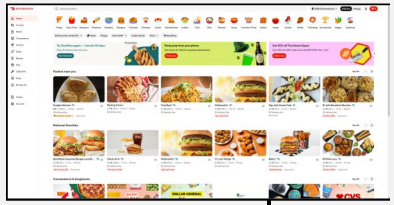
Problem Statement

Why do we need LUCI?

Embedded systems are becoming more powerful and increasingly integrated with **AI capabilities.**



Interaction with wider internet



Most embedded systems are optimized for speech input via lightweight ML-models

Most of user interfaces are optimized for touch or mouse input enabled by bigger screens.

OUR SOLUTION

Natural Language Interface

- Natural language-based commands parsed via text-to-speech interfaces.

Automated User Interface (UI) Orchestration

- Multi-step UI interface navigation to accomplish tasks such as ordering food.

On-device Runtime

- Enabled using <2B parameter LLMs which can run on device for low latency and no internet dependence.

Feature of LUCI

What makes LUCI tick.

1. Application-Centric planning

An application-centric planning framework for fast and efficient model grounding for adaptable multiapplication task planning.

2. OS - Agnostic

A modular OS-agnostic framework capable of scaling across native and web interfaces.

3. Efficient Rule-Based UI Representations

A rule-based semantic parser for efficient compression of front-end code into structured IAF representations, allowing for larger effective attention windows and better task grounding.

4. Multi-Agent framework

A multi-agent framework for task orchestration, enabling LUCI to achieve state-of-the-art performance on the Mind2Web benchmark while using lightweight LLMs

Limitations of Previous Approaches

What LUCI improves upon

LLM- Based Automation

API-Based Methods

These approaches navigate applications via APIs using the LLM's coding capabilities

Limitations

Poor Generalization

Due to API syntax variance, training is required for new applications

Context Length

Code is very verbose requiring large context windows

GUI-Based Methods

These approaches navigate applications via front-end code or large multi-modal LLMs with vision capabilities

Limitations

Low Information Density

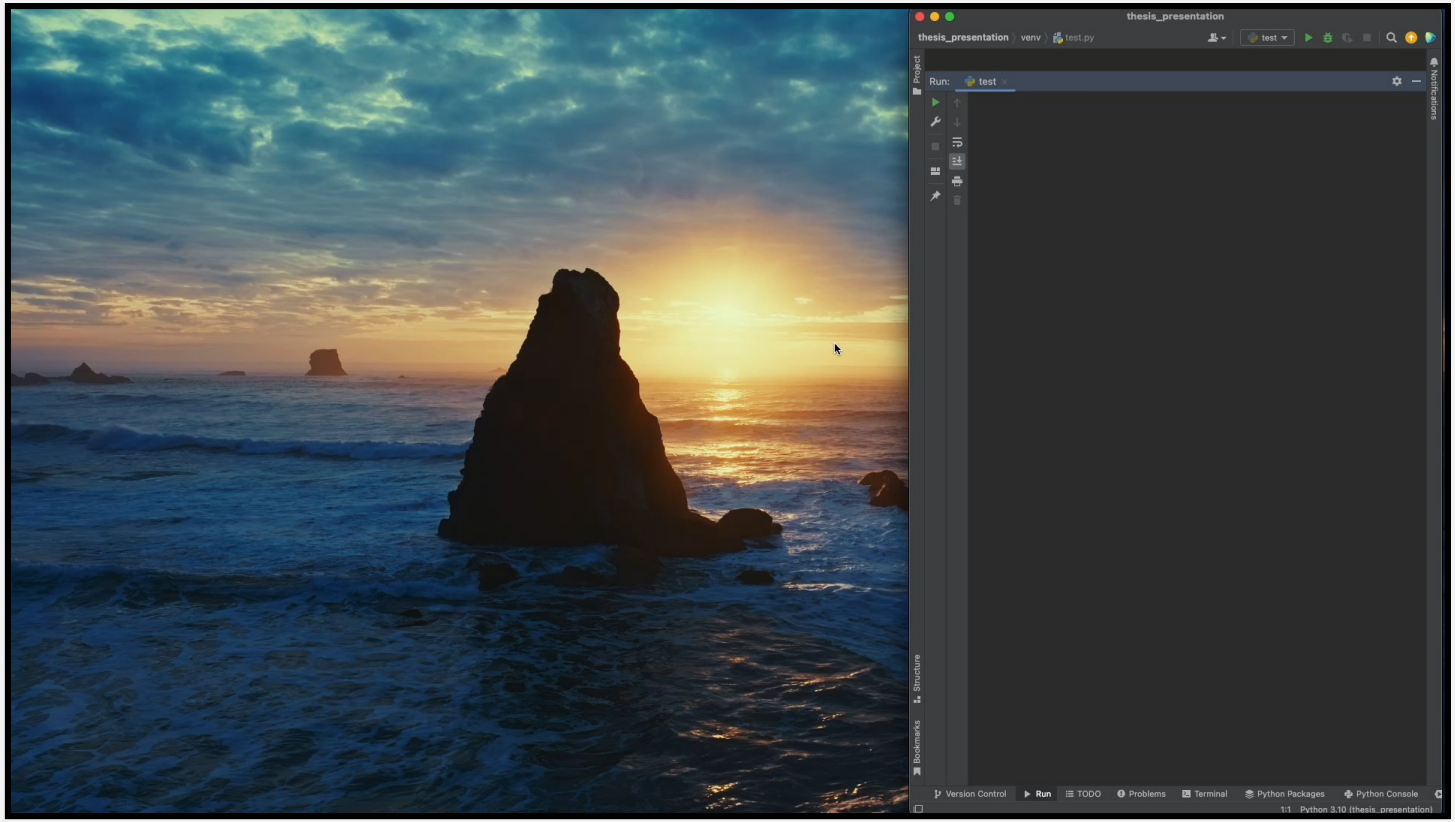
1. Application GUI code has a lot of excess information such as CSS styles.
2. When examined visually, websites have a lot of white-space and hence low information density.

Large Compute Requirement

1. Large amounts of redundant information in code requires a very large context window with creates a compute bottleneck
2. Multi-modal models require a much larger number of parameters and compute.

LUCI Demo

Seeing LUCI in Action



Demo of LUCI creating a power point presentation on recycling.

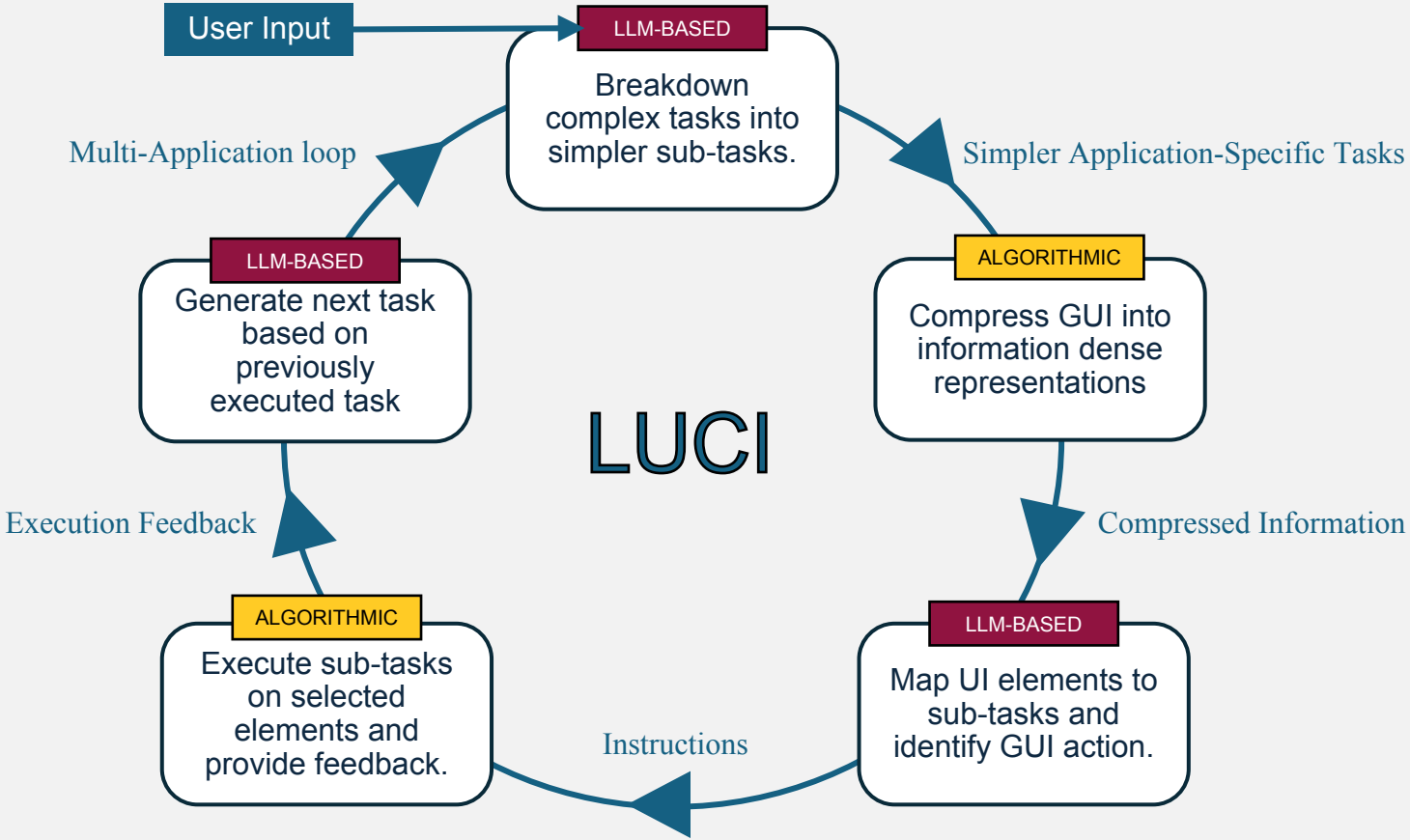
- LUCI is able to orchestrate tasks across multiple applications from a single command.
- LUCI can run locally due to using lightweight (<2B param) LLMs
- LUCI is able to dynamically generate content using the native generative capabilities of the LLMs

LUCI Architecture

Understanding the inner workings of LUCI.

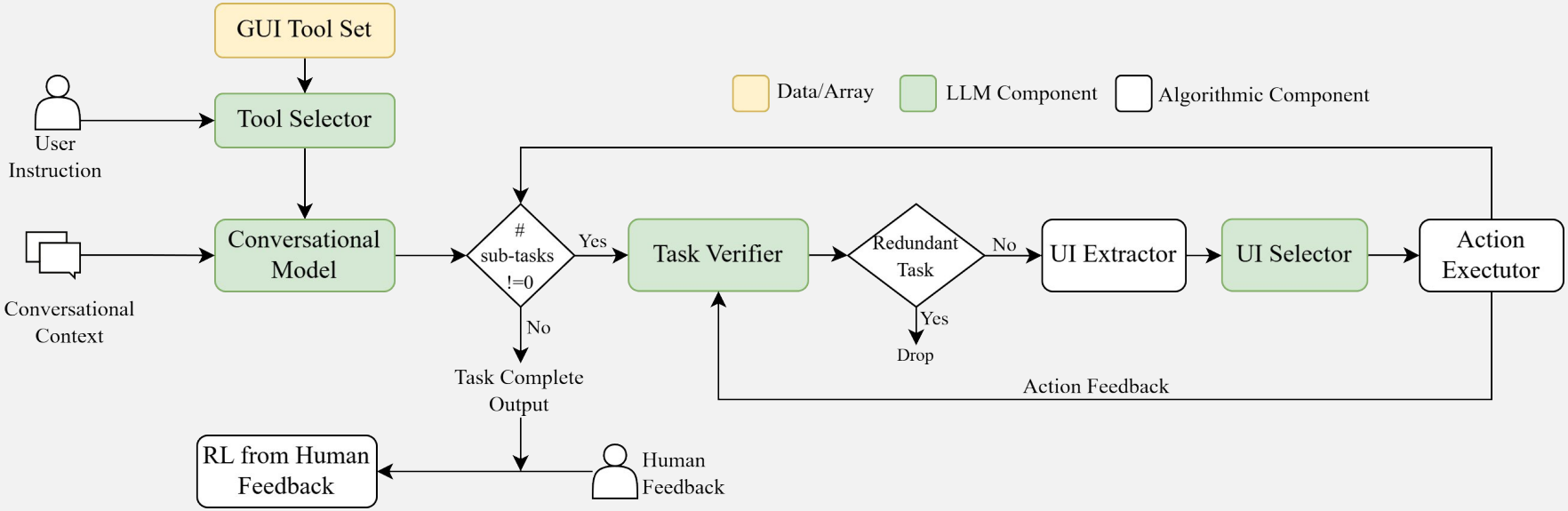
High level overview of LUCI System

How tasks are accomplished



LUCI Architecture

The components of LUCI and their relationships

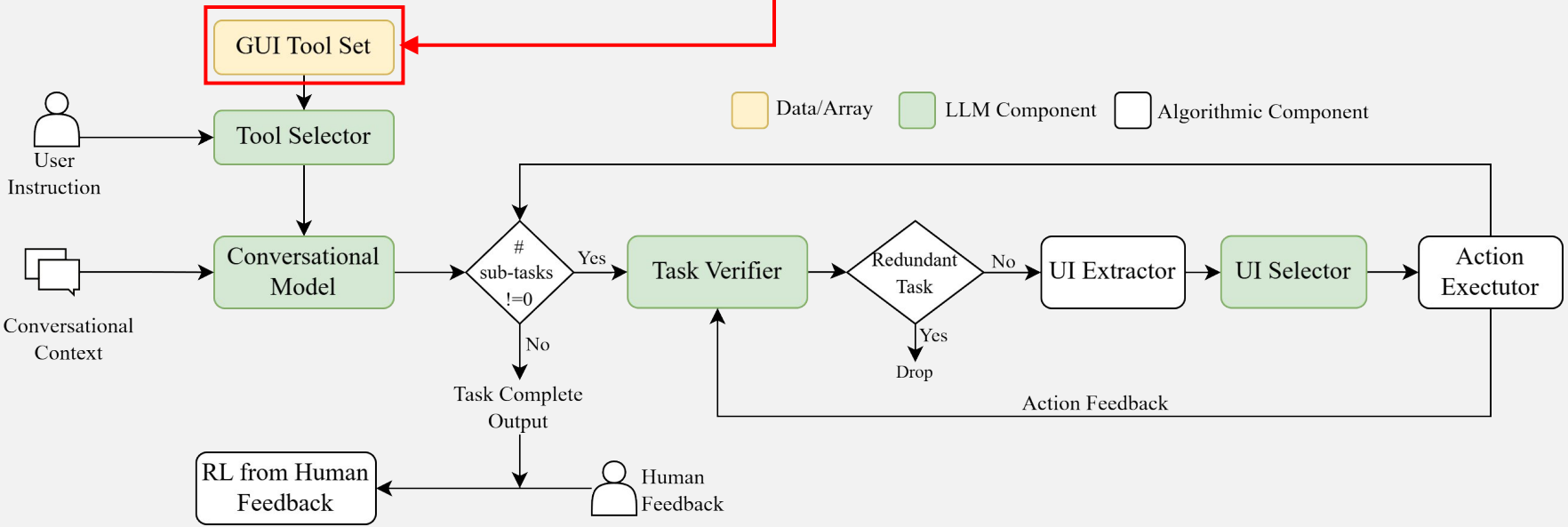


Architecture of LUCI: Given user instruction and the conversational context, the Tool Selector first selects a tool from the given GUI toolset and opens the applications. The conversational model then generates a solution outline, which is a text description of the list of sub-tasks needed to solve the task. Next, the Task Verifier filters redundant tasks in the solution outline based on action feedback from previously executed tasks and future sub-tasks. Then, a rule-based UI Extractor extracts UI elements from the GUI application. UI Selector selects appropriate UI elements from the list of UI elements generated by the UI extractor for the given sub-task. Lastly, the Action Executor performs an action on the selected UI element based on the type of UI element and generates the action feedback.

GUI Toolset

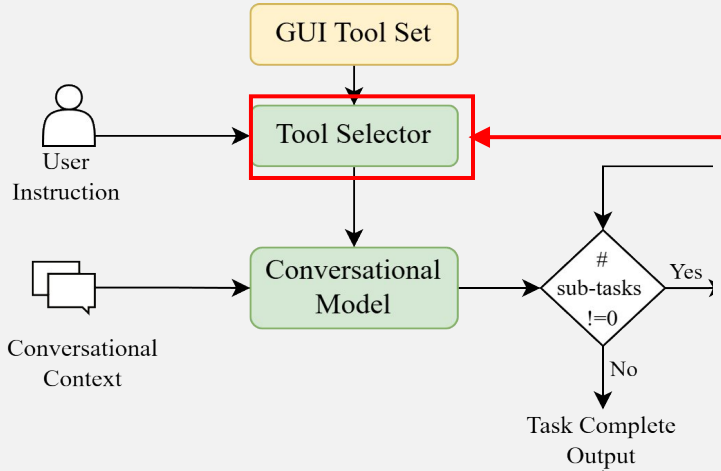
Tools at our disposal

- Contains a collection of different GUI application names that the agent is allowed to work on.
- LUCI has been tested with up to 22-Applications
- Web applications are accessed through browsers such as Safari, Chrome, etc.



Tool Selector

Application Centric Design



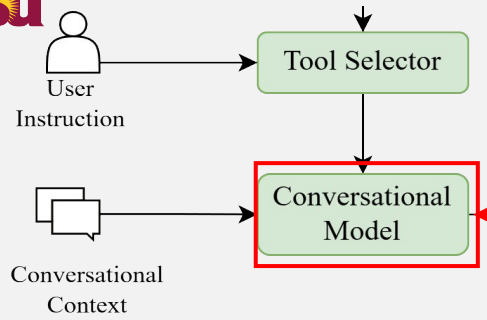
- The tool selector is an LLM based component which selects the appropriate Tool/Task based on the subtask.
- **Input:** <User Instruction, GUI Tool set>,
  Output: Desktop Application Name

Example Prompt

GUI Tool Set (G) : Calculator, Calendar, Keynote, Maps, Messages, Music, News, Numbers, Pages, Safari, Stocks, TextEdit, Weather, Script Editor, Terminal, Outlook.
Example Query 2: From the given list of GUI tools G, select a tool to solve the task "Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn? And write the entire proof."
Response: Calculator, Text Edit

Conversational Model

Planning … one task at a time.



- Generates sub-tasks from user instruction and past conversation using Chain-of-Thought reasoning
- **Input:** <Desktop Application name, User Instruction, Conversation Context>, **Output:** Solution Outline

Example Prompt

Example Query 5: What are the tasks to perform on Calculator and Text Edit for the given task "Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn? And write the entire proof". Generate the tasks in the format "Tool Required: True / False, Tool: tool Name, Task description: description of the task".

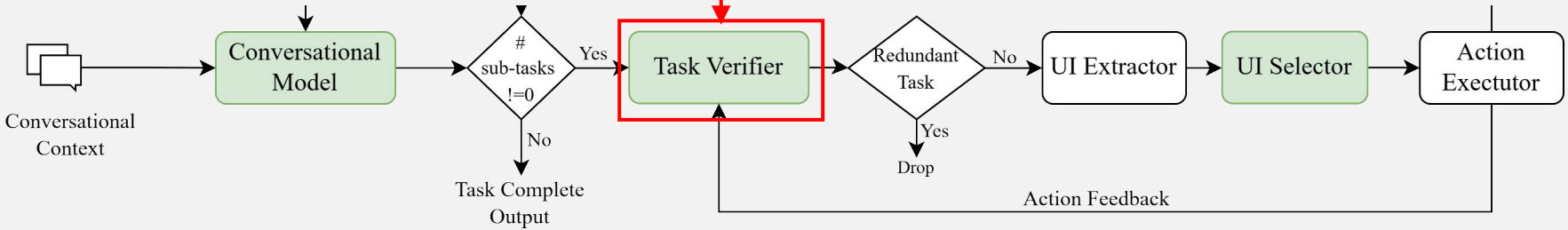
Response:

1. Tool Required: False, Tool: Calculator, Task description: Open Calculator on your Macbook.
2. Tool Required: True, Tool: Calculator, Task description: Set the calculator to the "Standard" mode".
3. Tool Required: True, Tool: Calculator, Task description: Convert 50 minutes to hour by dividing 50 by 60 in the Calculator.
4. Tool Required: True, Tool: Calculator, Task description: Press the equals (=) key to calculate the result.
5. Tool Required: True, Tool: Calculator, Task description: Multiply number of minutes with money earned per hour ($12)
6. into the Calculator.
7. Tool Required: True, Tool: Calculator, Task description: Press the equals (=) key to calculate the result.
8. Tool Required: True, Tool: Calculator, Task description: Observe the calculated result displayed on the Calculator screen.
9. Tool Required: False, Tool: Text Edit, Task description: Open Text Edit.
10. Tool Required: False, Tool: Text Edit, Task description: Generate a summary of the working.
11. Tool Required: True, Tool: Text Edit, Task description: Write the summary to Text Edit.

Task Verifier

Improving Efficiency

- Prior to sub-task execution, redundant tasks are eliminated to improve efficiency based on if the future tasks and if it was already executed in a prior step
- **Input:** <Current Sub-task, Future Sub-tasks, Action feedback>
  Output: <Reason to execute, Execute yes / no>



Example Prompt

Example Query 6 : Check whether the current task s3 is necessary for the objective I based on previously executed tasks {(sl; AFI), (s2; AF2)} and future tasks s4, s5, s6, s7. The answer format should be " Reason: Reason to execute sub-task , Answer: Yes/No".
Response: Reason: This task is required to convert 50 minutes to hours, Answer: Yes

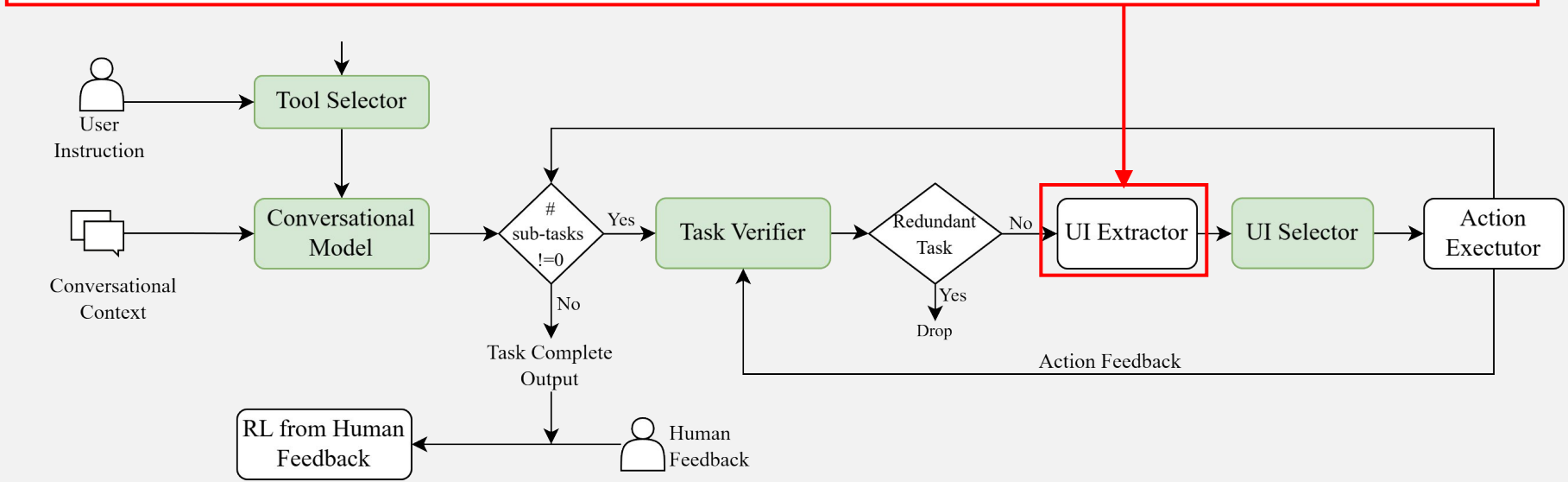
UI Extractor

Application Centric Design

- In most applications, the front-end code is based on a tree-structure. We parse through tree structure using a bottom-top approach creating an intermediate representation of the UI dubbed the Information-Action-Field (IAF) representation.
- The front-end is then recursively parsed into an IAF representation -

‘**I [I1, < *A*1, *F*1 >, *I*2, < *A*2, *F*2 >, *I*3, < *A*3, *F*3 >, …]**’,

Where, I1,I2,I3 contain all the information about a node (ex. Heading) and <An, Fn> are the actions with their associated fields (ex. Form and submit button)



UI Element Categories

Semantic Divisions in GUI

Information Elements (I) : UI elements that contain only Information.
Example: **< p >, < h1> … < h6>, , < div >, etc.**

Field Elements (F) : : UI elements that collect user input.
Example**: textbox, checkbox, radio buttons, etc.**

Action Elements (A): UI elements which perform post and get methods.
Example: **buttons, hyperlinks, etc.**

UI Parsing Assumptions

Semantic Divisions in GUI

Based on the categorization into Information, Action and Field. The following assumptions are made when parsing the GUI.

Input POST Assumption

- User input is sent to server when a post method is called. It means every F is associated with a A. For instance the submit button comes after the search box

Field-Action Form Assumption

- If there are multiple field nodes when parsing a branch, they are all associated to the same action node.
- Each field must be associated to an action

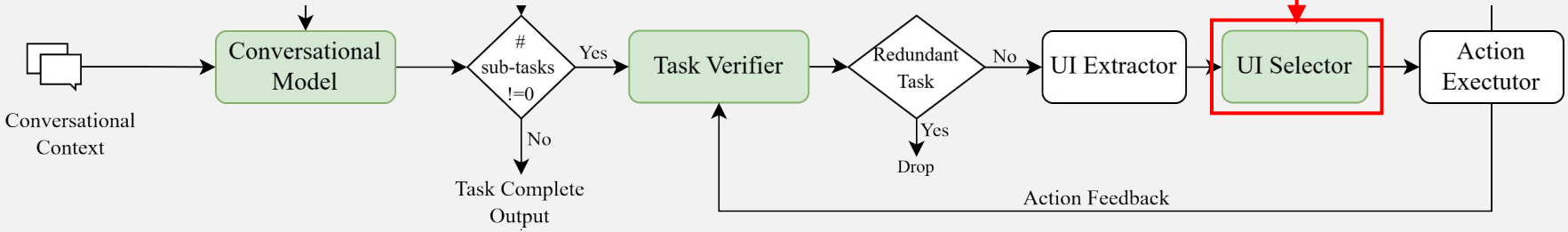
Information-Action Assumption

- If information nodes have an associated action node, they are either the child node or sibling node of the information node.

UI Selector

Selection actionable items

- An LLM selects UI elements from a list of possible elements based on the sub-task, therefore grounding the model.
- **Input:** <Sub-task description, list of UI elements>
  Output: UI elements



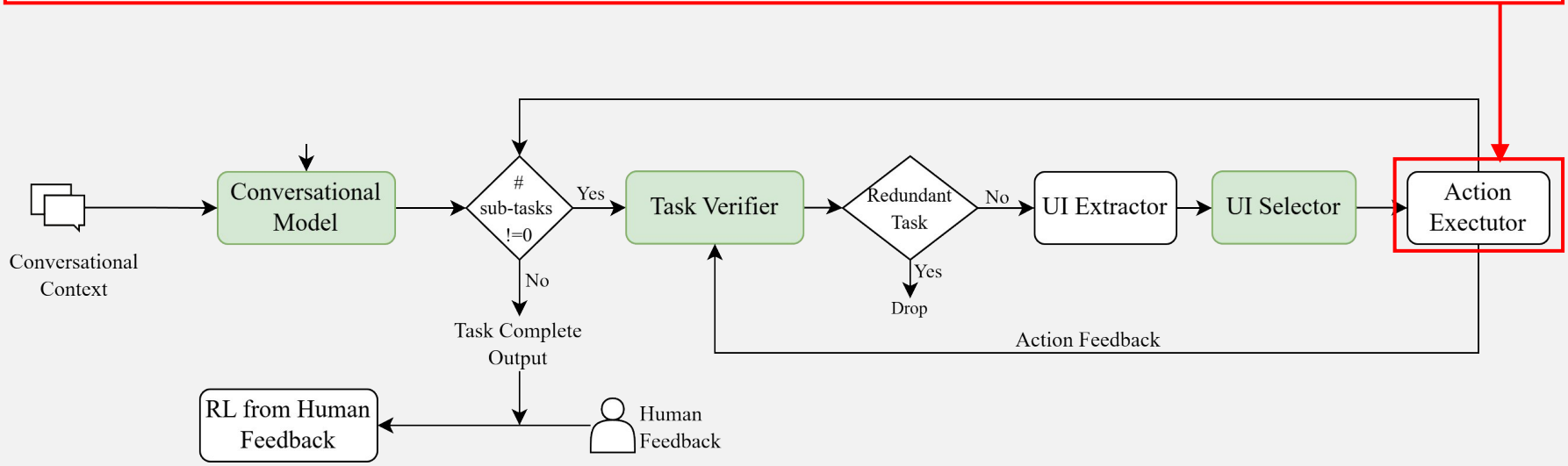
Example Prompt

UI elements (U): For Calculator available UI elements are,
window : {
1, 2, 3, 4, 5, 6, 7, 8, 9, 0, =, +, , ×, ÷, ., +/, %, all clear, close button, zoom button, minimize button, main display: {0}
}
Menu bar : {
File: {Close, Close All, Save Tape As, Page Setup, Print Tape},
Edit: {Undo, Redo, Cut, Copy, Paste, Clear, Select All, Start Dictation, Emoji Symbols},
View: {Basic, Scientific, Programmer, Show Thousands Separators, RPN Mode, Decimal Places, Enter Full Screen}
Convert: {Recent Conversions, Area, Currency, Energy or Work, Length, Power, Pressure, Speed, Temperature, Time, Volume, Weights and Masses}
}
Example Query 7 : From the given list of available UI elements : U, select list of UI elements required for task "Convert 50 minutes to hour by dividing 50 with 60 in the Calculator" . The answer should be format [UI element 1, UI element 2, ...]
Response: [5, 0, ÷, 6, 0, =]

Action Executor

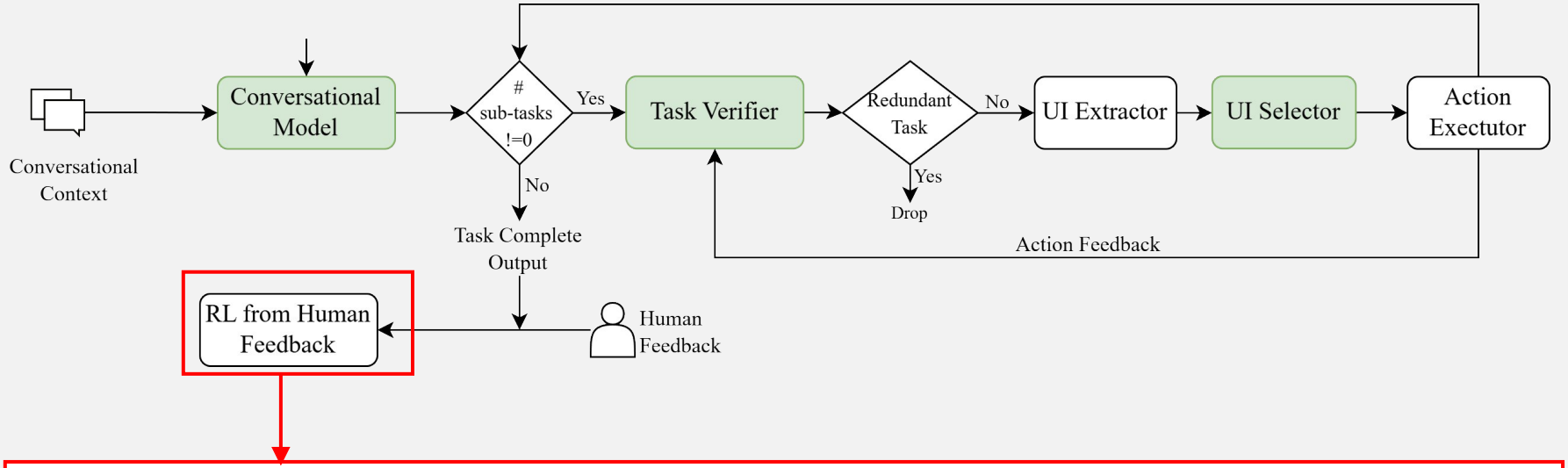
Words to Action

- A set of valid actions are generated based on the UI elements (ex. '*click*' for buttons)
- A feedback mechanism -- check if an action was executed or not -- to provide feedback for future tasks.
- Upon the completion of all sub-tasks, a signal is sent to the conversation model to generate execution feedback and return the results of the query to the user.
- **Input:** UI Elements
  Output: Action Feedback



Learning from Feedback

Improving as we go



- RLHF learning mechanism is used to improve the task planning and task prioritization by the controller.
- Human feedback is combined with execution feedback, this combined feedback is denoted by F.

Experiments and Results

LUCI in practice

Mind2Web Performance

Benchmark performance

- Evaluated LUCI's generalizability over web tasks and compared with baselines.
- The dataset is divided into three test sets: Cross-Task, Cross-Website, and Cross-Domain, evaluating generalizability over tasks from the same, similar and completely unseen domains, respectively.
- Evaluation Metrics :
  1. **Operation F1** (Op. F1) for token-level F1 score for predicted operation comprised of action and input value
  2. **Step Success Rate** for success rate per task step.
  3. **Success Rate** for successfully executing the entire task.

Mind2Web Performance

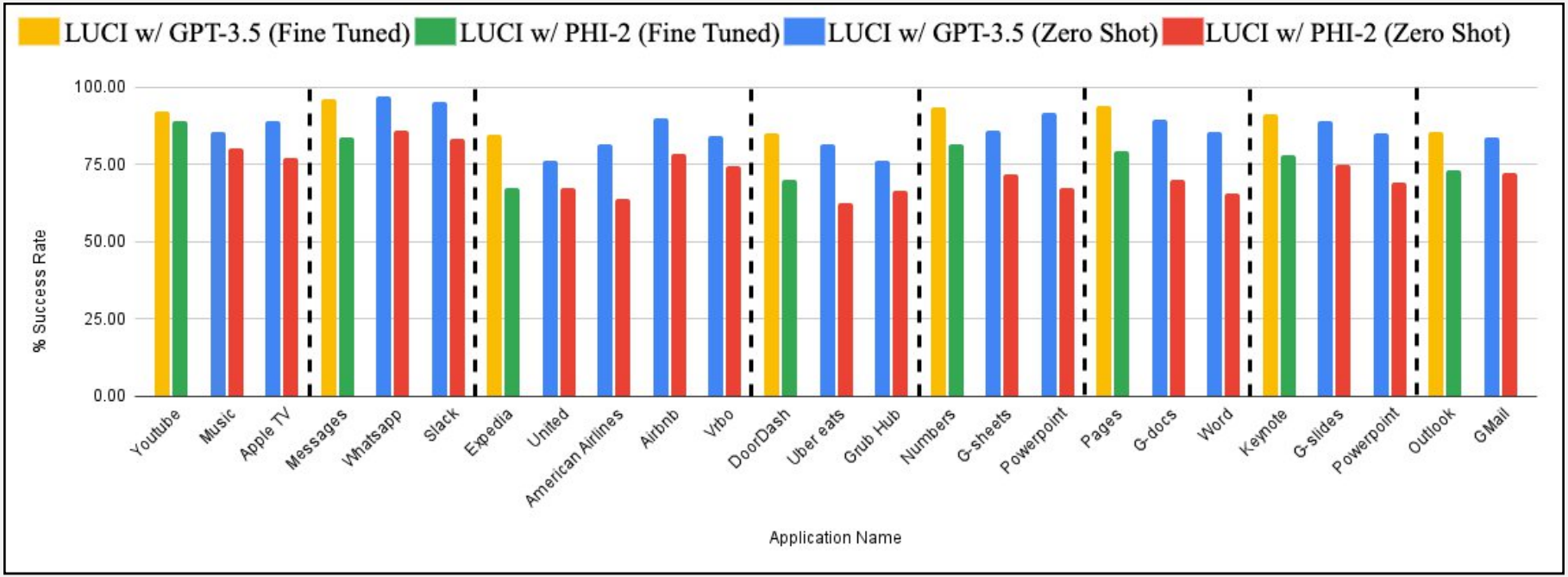
Benchmark performance

Baseline	Cross - Task			Cross - Website			Cross - Domain		
	Op. F1	Step SR	SR	Op. F1	Step SR	SR	Op. F1	Step SR	SR
MindAct	56.6	17.4	0.8	48.8	16.2	0.6	52.8	18.6	1.0
Synapse	-	30.6	2.4	-	29.1	0.6	-	26.4	1.5
WebGUM	75.9	64.9	-	75.3	62.5	-	77.7	66.7	-
GPT-4V	80.9	65.7	50	83.7	70	-	73.6	62.1	-
LUCI w/GPT-3.5	**93.8**	**86.7**	**76.4**	**96.3**	**89.1**	**81.9**	**91.7**	**84.2**	**74.2**
LUCI w/PHI-2	**82.3**	**82.8**	**66.8**	**84.9**	**83.3**	**70.5**	**79.4**	**79.1**	**61.3**

Average performance on Mind2WEB Benchmark. LUCI w/ PHI-2 is competitive with if not better than other heavier methods, which using GPT 3.5, LUCI is significantly better than state of the art.

Cross-Application Adaptability

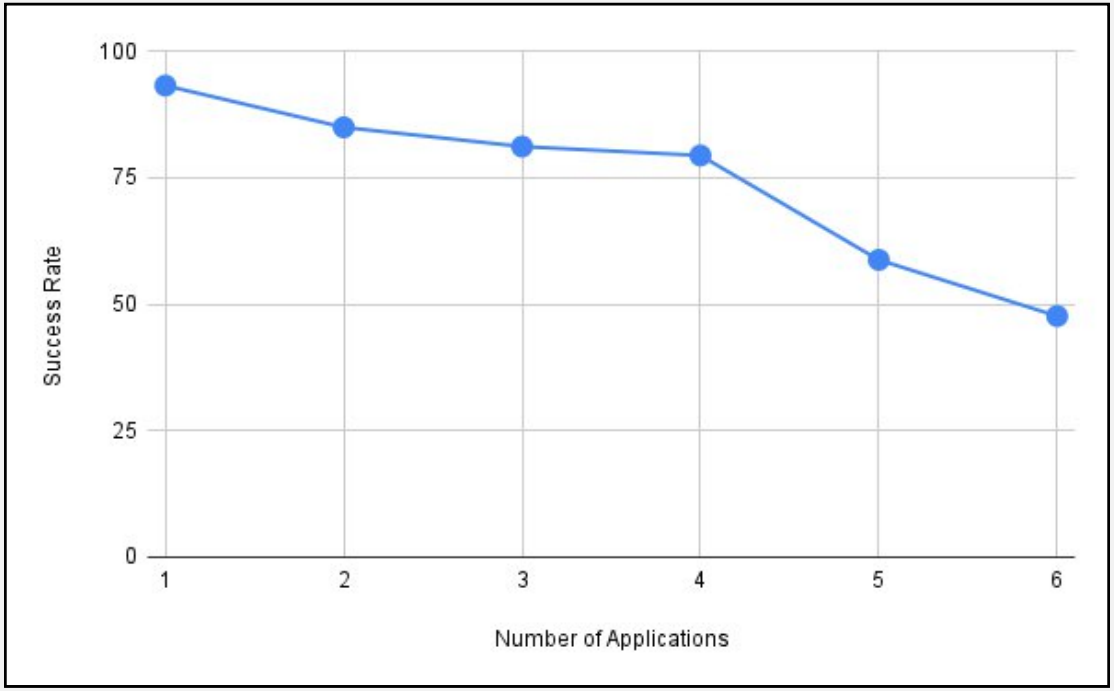
Benchmark performance



Cross application performance of LUCI with GPT-3.5 and PHI-2: LUCI fine-tuned on an application that exhibits comparable performance on similar unseen applications.

Scaling to Multiple Applications

LUCI context tests across applications



Average success rate of LUCI across tasks involving the use of multiple applications. The trend shows LUCI's ability to use at least four applications without losing efficacy.

Summary

LUCI context tests across applications

- Natural language-based Interface desirable in Embedded Systems
- LUCI: Lightweight UI Command Interface
  - Application centric planning
  - OS-Agnostic
  - Multi-agent framework
  - Efficient rule-based UI represent
- Given user instruction and the conversational context:
  - the Tool Selector first selects a tool from the given GUI toolset and opens the applications.
  - The conversational model then generates a solution outline, which is a text description of the list of sub-tasks needed to solve the task.
  - Next, the Task Verifier filters redundant tasks in the solution outline based on action feedback from previously executed tasks and future sub-tasks.
  - Then, a rule-based UI Extractor extracts UI elements from the GUI application.
  - UI Selector selects appropriate UI elements from the list of UI elements generated by the UI extractor for the given sub-task.
  - Lastly, the Action Executor performs an action on the selected UI element based on the type of UI element and generates the action feedback.
- LUCI Results
  - LUCI w/ PHI-2 are competitive if not better than other heavier methods.

24