Multimodal Applications

CS 224V
Monica Lam, Jackie (Junrui) Yang
Lecture Goals

• Why do we need multimodal interactions?

• Introduction to existing multimodal interactions

• Three problems for multimodal app development

• Introduction to ReactGenie: a multimodal app development framework

• Evaluation of ReactGenie
Lecture Goals

• Why do we need multimodal interactions?
• Introduction to existing multimodal interactions
• Three problems for multimodal app development
• Introduction to ReactGenie: a multimodal app development framework
• Evaluation of ReactGenie
Why Multimodal Interaction

• Limitations of voice assistants
  • Lack of discovery
  • Slow for long text
  • Cannot show graphics
  • Interactive analog adjustments
  • Can we combine the best of both words?

• Advantages of GUI
  • Explore the possible functions in the menus
  • Glance over long lists and paragraphs
  • Images, videos, PowerPoint
  • CAD tools, PhotoShop, WYSIWYG
Example: Powerpoint

• Lots and lots of nested menus
Example: Powerpoint

- Lots and lots of nested menus
- Horizontal ones and vertical ones
- Takes a long time to make slides (even if you know what functions are available)
Here are Some Examples:

• Make this text box bold in the slide master.

• Make the border of this shape with little dots.

• Make everything right aligned on this slide.

• Make every shape on this slide above this yellow.
Solution: Multimodal Applications

• User: Much faster if we can give the high-level intent

  • Don’t have to figure out which menus to open and where the features are listed
  • Don’t have to learn the exact feature names
  • Don’t have to do repetitive work on each slide/bullet point, etc.
Lecture Goals

• Why do we need multimodal interactions?

• **Introduction to existing multimodal interactions**

• Three problems for multimodal app development

• Introduction to ReactGenie: a multimodal app development framework

• Evaluation of ReactGenie
Multimodal Interaction: Voice + Gesture

Put That There
November 2, 1979
The Architecture Machine
© 1979 MIT

1982 Bolt et al.
Multimodal Interaction: Voice + Gesture

QuickSet
1997 Cohen et al.
Multimodal interfaces includes:

- Input
  - Visual (Face location, Gaze, Facial expression, Lipreading, Sign language)
  - Auditory (Speech input)
  - Touch (Pressure, Location and selection, Gesture)
  - Other sensors (Sensor-based motion capture)
Multimodal input: Gaze + Gesture

Example Applications

Gaze + Gesture
2015 Xiao et al.
Multimodal interfaces includes:

- **Input**
  - Visual (Face location, Gaze, Facial expression, Lipreading, Sign language)
  - Auditory (Speech input)
  - Touch (Pressure, Location and selection, Gesture)
  - Other sensors (Sensor-based motion capture)

- **Output**
  - Graphics (Lists, Buttons, Charts, Pictures)
  - Sound (Speech, Music, Sound Effects)
  - Touch (Haptics)
  - Other outputs (Temperatures, AR/VR)
Multimodal Output: Graphics + Sound

OLGA

1998 Sundblad et al.
Multimodal interfaces can (Oviatt 2000):

• Flexibility: Permit combination and alternating input modes.
• Efficiency: Support effective manipulation, especially graphical information.
• Clarity: Simplify and improve speech interfaces.
• Precision: Offer accurate spatial information.
• Options: Provide multiple interaction techniques.
• Error Reduction: Enhance error avoidance and resolution.
• Inclusivity: Accommodate diverse users and tasks.
• Adaptability: Resilient to changing environmental conditions.
• Accessibility: Cater to individual needs, including handicaps.
• Balance: Help prevent overuse of any single mode.
Multimodal interfaces can (Oviatt 2000):

- Flexibility: Permit combination and alternating input modes.
- Efficiency: Support effective manipulation, especially graphical information.
- Clarity: Simplify and improve speech interfaces.
- Error Reduction: Enhance error avoidance and resolution.
Flexibility of Multimodal Interfaces

*i*Chameleon

A Scalable and Extensible Framework for Multimodal Interaction

Will W.W. Tang  Kenneth W.K. Lo
Alvin T.S. Chan  Stephen C.F. Chan
Hong Va Leong  Grace Ngai
Dept of Computing, Hong Kong Polytechnic Univ

2013 Tang et al.
Efficiency of Multimodal Interfaces

PixelTone

A Multimodal Interface for Image Editing

Everyone takes pictures, but photo editing can be hard

PixelTone

2013 Laput et al.
Clarity of Multimodal Interfaces

ReMap: Lowering the Barrier to Help-Seeking with Multimodal Search

C. Ailie Fraser, Julia M. Markel, N. James Basa, Mira Dontcheva, Scott Klemmer

ReMap
2020 Fraser et al.
Error Reduction of Multimodal Interfaces

Voice and Touch Based Error-tolerant Multimodal Text Editing and Correction for Smartphones

Maozheng Zhao¹, Wenzhe Cui¹, I.V. Ramakrishnan³, Shumin Zhai², Xiaojun Bi¹

¹Stony Brook University, ²Google LLC

2021 Zhao et al.
Lecture Goals

• Why do we need multimodal interactions?
• Introduction to existing multimodal interactions
• Three problems for multimodal app development
  • Compositionality of multimodal commands
  • Expose diverse actions/APIs from a GUI app
  • Allows for interchangeable and simultaneous multimodal interactions
• Introduction to ReactGenie: a multimodal app development framework
• Evaluation of ReactGenie
Recall the examples earlier

• Make this text box bold in the slide master.

• Make the border of this shape with little dots.

• Make everything right aligned on this slide.

• Make every shape on this slide above this yellow.
Problem 1: Compositionality of multimodal commands

- Make this text box bold in the slide master.
  
  `MakeSlideMasterTextBold()`

- Make the border of this shape with little dots.
  
  `MakeShapeBorder(borderType:"littleDots")`

- Make everything right aligned on this slide.
  
  `SetEverythingAlignment(alignment:"Right")`

- Make every shape on this slide above this yellow.
  
  `SetShapeAboveColor(color:"Yellow")`

This is not feasible with simple function calling!!!
Compositionally of multimodal commands

• Make this text box bold in the slide master.


• Make the border of this shape with little dots.

Shape.Current().lineFormat.setDashStyle(dashStyle:"RoundDot")

• Make everything right aligned on this slide.

Slide.Current().getShapes().textFrame.textRange.paragraphFormat.setHorizontalAlignment(horizontalAlignment:"Right")

• Make every shape on this slide above this yellow.

Slide.Current().getShapes().between(field:.top,to:Shape.Current().top).fill.setForeGroundColor(color:"yellow")
Solution to compositionally: ReactGenieDSL

- Python:
  ```python
  Slide.Current().findShape(
    above=Shape.Current())
  Weak type, more errors
  ```

- TypeScript:
  ```javascript
  Slide.Current().findShape(
    Shape.Current().top
  ).forEach((x)=> x.delete())
  No param names, more errors
  ```

- Swift:
  ```swift
  Slide.Current().findShape(
    above: Shape.Current().top
  ).forEach{$0.delete()}
  Ambiguous query
  ```

- SQL:
  ```sql
  select * from shape
  where top>current_top
  No Action
  ```

- Existing language:
  - Weak type, more errors
  - No param names, more errors
  - Ambiguous query
  - No Action

- ReactGenieDSL:
  - Easy to generate
  - Versatile query
  - Fewer errors
  - No Lambda Expression

- Automatically distributed to each element
# Expressiveness of ReactGenieDSL

In a series of apps

<table>
<thead>
<tr>
<th>Feature of RGDSL</th>
<th>English grammar</th>
<th>Food Ordering</th>
<th>PowerPoint</th>
<th>Social Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribute action to an array of objects</td>
<td>Plural Object + Verb</td>
<td>Order.All()[-1].foods.like()</td>
<td>Slide.Current().getShapes().textFrame.setText(&quot;&quot;)</td>
<td>User.Me().posts.delete()</td>
</tr>
<tr>
<td>Select objects to do actions</td>
<td>Object Modifier + Object + Verb</td>
<td>Restaurant.Current().foods.sort(field: .price)[0].order()</td>
<td>Shape.All().matching(field: .textFrame.text, value: &quot;yellow&quot;).delete()</td>
<td>Post.All().equals(field: .like, value: true)</td>
</tr>
</tbody>
</table>
ReactGenie System Architecture

- Semantic Parser
- Runtime
- UI Mapping
- Response Generator
- UI Update
Genie PowerPoint

- 110 APIs: Slide, Shape, SlideMaster, TextFrame, TextRange, ...
  - That’s everything from MS PowerPoint JS API.
  - Yet, it still does not have common APIs like Animation, Font color, etc.
  - It is super hard to expose API outside of the development cycle of GUI
  - With the advancement of AI, developers need to build two interfaces:
    - A human interface, and
    - An AI interface
  - That’s double the work on developers!
Problem 2: Expose diverse actions/APIs from a GUI app

- 110 APIs: Slide, Shape, SlideMaster, TextFrame, TextRange, ...
  - That’s everything from MS PowerPoint JS API.
  - Yet, it still does not have common APIs like Animation, Font color, etc.
- It is super hard to expose API outside of the development cycle of GUI
- With the advancement of AI, developers need to build two interfaces:
  - A human interface, and
  - An AI interface
- That’s double the work on developers!
Problem 2: Expose diverse actions/APIs from a GUI app

```java
@GenieClass("Past order or a shopping cart")
class Order extends DataClass {
  @GenieKey()
  public orderId: string;
  @GenieProperty("Items in the order")
  public orderItems: FoodItem[];
  constructor({orderId, orderItems}: {orderId: string, orderItems: FoodItem[]}) {
    super({orderId, orderItems});
    this.orderId = orderId;
    this.orderItems = orderItems;
  }
  @GenieFunction()
  static All(): Order[] {
    return fetchOrdersFromServer();
  }
  @GenieFunction("Create a new order")
  static CreateOrder(): Order {
    return new Order({orderId: randomId(), orderItems: []});
  }
  @GenieFunction("Add an item to the order")
  addItem({foodItem}: {foodItem: FoodItem}) {
    this.orderItems.push(foodItem);
    updateServer();
  }
}
```

React App Logic Code
When using a ReactGenie app

Which restaurant?

How can the user know what happened?
Problem 3: Allows for interchangeable and simultaneous multimodal interactions

UI input mapping

UI updates

Navigation
Problem 3: Allows for interchangeable and simultaneous multimodal interactions

• Multimodal Input
  • UI Input Mapping: Getting what object that I’m touching

• Multimodal Output
  • Resulting objects are on-screen — UI Updates: Update the values on screen
  • Resulting objects are off-screen — Navigation: Navigate to the page with results
Lecture Goals

• Why do we need multimodal interactions?

• Introduction to existing multimodal interactions?

• Three problems for multimodal app development

• Introduction to ReactGenie: a multimodal app development framework

• Evaluation of ReactGenie
What is ReactGenie?

Ease of Development

React Code
(In recommended style) + Annotations
(< 5% code typically) → ReactGenie Code

Rich Multimodal Functionality

Combinations of Voice + GUI input → Semantic Parser
UI Mapping → ReactGenie DSL Code → Execution → Both Voice + GUI output
ReactGenie Demo
Modern GUI 101: State + Components

State Code: Implements Features

```javascript
class Recipe {
  name: String;
  img: Image;
  loved: boolean;

  love(): void {
    this.loved = true;
  }
}
```

Components: Describe GUI

```javascript
RecipeViewImpl = (recipe: Recipe) => {
  return (
    <div>
      <img image={recipe.img} />
      <love loved={recipe.loved}
      onClick={()=>recipe.love()}/>
      <div> {recipe.name} </div>
    </div>
  )
}
```
ReactGenie = React + Annotations

RecipeViewImpl = (recipe: Recipe) => {
  return (
    <div>
      <img image={recipe.img} />
      <love loved={recipe.loved} onClick={()=>recipe.love()}/>
      <div> {recipe.name} </div>
    </div>
  )
}

RecipeView = GenieInterface("Recipe", RecipeViewImpl)

@DataClass()
class Recipe: GenieClass {
  @GenieProperty()
  name: String;
  img: Image;
  @GenieProperty()
  loved: boolean;

  @GenieFunction()
  love(): void {
    this.loved = true;
  }
}

State Annotations:
Which class/property/function can be accessed with voice

Components Annotations:
Which components represent which state classes
Recall the Agent Architecture (No Policies)

Agent

Text input → Semantic Parser → User Dialogue State

Execution

Result → Response Generator → Agent Output

Dialogue State
React has a Similar Architecture

**Agent**

- Text input → Semantic Parser → Execution → Result → Response Generator → Agent Output
  - Dialogue State

**React**

- Text input → UI Mapping → Execution → Result → UI Update → Display
  - Component API call
  - React Program State

- Execution of React Programs
  - Updates the state of the variables
  - Navigates to a display page (all current values are displayed)
Multimodal Agent Architecture

ReactGenie

- Execution of ReactGenie Programs
  - Updates the state of the variables
  - Navigates to a display page (all current values are displayed)
  - Generates the text result
Semantic Parser + Response Generator

ReactGenie

Text input

Semantic Parser

User Dialogue State (ReactGenieDSL)

Execution

React Program State

Response Generator

Agent Output

Text input

UI Mapping

Component API call

UI Update

Display
Revisiting ReactGenieDSL

• Make this text box bold in the slide master.
  

• Make the border of this shape with little dots.
  
  `Shape.Current().lineFormat.setDashStyle(dashStyle:"RoundDot")`

• Make everything right aligned on this slide.
  
  `Slide.Current().getShapes().textField.textRange.paragraphFormat.setHorizontalAlignment(horizontalAlignment:"Right")`

• Make every shape on this slide above this yellow.
  
  `Slide.Current().getShapes().between(field:.top,to:Shape.Current().top).fill.setForeGroundColor(color:"yellow")`
### LLM-Based Semantic Parser

Here are all the functions that we have:

```java
class Restaurant {
    string name;
    string address;
    string cuisine;
    float rating;

    // All active restaurants
    static Restaurant[] all();

    // The current restaurants
    static Restaurant current();

    // Get a list of foods representing the menu from a restaurant
    Food[] menu;

    // Book reservations on date
    Reservation get_reservation(date: DateTime)
}
```

#### Examples:

- **user**: get me the best restaurant in palo alto
  - **agent**: `Restaurant.all().matching(field: .address, value: "palo alto")`

#### User interaction

- **user**: order the same burger that I ordered at mcDonald last time
  - **parsed**: `Order.current.addFoods(foods: Order.all().matching...)`

---

### Declaration

### Few-shot examples

### Current interaction

### Parsed result
// Here are all the functions that we have

class Restaurant {
  string name;
  string address;
  string cuisine;
  float rating;

  // All active restaurants
  static Restaurant[] all();

  // The current restaurants
  static Restaurant current();

  // Get a list of foods representing the menu from a restaurant
  Food[] menu;

  // Book reservations on date
  Reservation get_reservation(date: DateTime)
}

// Generate concise voice feedback for the user's command

// User interaction
user: order the same burger that I ordered at mcDonald last time
parsed: Order.current.addFoods(foods: Order.all().matching...
execution_result: {
  "type": "Order",
  "items": [{
    "type": "FoodItem",
    "name": "Hamburger"
  }, {
    "type": "FoodItem",
    "name": "Fries"
  }]
}

response: Your order with a hamburger and fries has been placed.
How to Handle Hybrid Inputs

ReactGenie

Text input → Semantic Parser → User Dialogue State (ReactGenieDSL) → Execution

Text input → UI Mapping → Component API call → Execution

Result → Response Generator → Agent Output

Result → UI Update → Display
UI Mapping

Programming Objects

Restaurant (name: “McDonald”)
Order (date: “3/3/2023”)
FoodItem (name: “boba tea”)
FoodItem (name: “mango drink”)

View
ReactGenie
How to Handle Hybrid Inputs

User: “Reorder this food”

Order.GetActiveOrder().addFood(food:[FoodItem.Current()])

Unresolved UI reference FoodItem.Current()

UI Mapping: Find FoodItem closest to the click point

FoodItem(name:"CrunchWrap")

Continue execution
How to Navigate with Voice Commands

ReactGenie

Text input → Semantic Parser

Text input → UI Mapping

User Dialogue State (ReactGenieDSL)

Execution

Component API call

React Program State

Result → Response Generator → Agent Output

Result → UI Update → Display
UI Update 1: Object is on UI → React takes care of it

“"I love the Creamy Potatoes recipe!"

Recipe(name: “Creamy”). love()

recipe.loved = true

RecipeViewImpl = (recipe: Recipe) => {
  return (
    <div>
      <img image={recipe.img} />
      <love loved={recipe.loved} onClick(()=>recipe.love())/>
      <div> {recipe.name} </div>
    </div>
  )
}
UI Update 2: Object not on UI → Navigate to the page

“Show me the Creamy Potato recipe!”

Recipe(name: “Creamy Potato”)
Execution

ReactGenie

Text input → Semantic Parser → User Dialogue State (ReactGenieDSL) → Execution

Text input → UI Mapping → Component API call → Execution

Execution → Result → Response Generator → Agent Output

Execution → React Program State → UI Update → Display
"Make everything above this yellow"

```swift
Slide.Current().getShapes().between(field:.top, to: Shape.Current().top).fill.setForeGroundColor(color: "yellow")
```

- Slide.Current()
  - Slide(id: 1)
  - Slide.Current().getShapes()
    - [Shape(text: "A"), Shape(text: "B"), Shape(text: "C"), Shape(text: "D"), Shape(text: "E")]
- Slide.Current().getShapes().between(field:.top, to: Shape.Current().top)
  - 100
- Slide.Current().getShapes().between(field:.top, to: Shape.Current().top)
  - [Shape(text: "A"), Shape(text: "B")]
- Slide.Current().getShapes().between(field:.top, to: Shape.Current().top).fill.setForeGroundColor(color: "yellow")
  - [Fill(), Fill()]
```
ReactGenie = React + Annotations

State Annotations:
Which class/property/function can be accessed with voice

Components Annotations:
Which components represent which state classes
Recap: ReactGenie Uses Declarative UI Architecture for Ease of Development

Define Object-Oriented States
- Taco Bell
- Taco 3/3
- Mr Sun 3/3
- Crunchwrap
- Quesadilla
- Taro boba

Define UI Components
- Restaurant ItemView
- OrderItem View
- Food Thumbnail

Home
- Featured Restaurants
- Featured Order History
Recap: ReactGenie Use Annotations for Multimodal Integration

```java
@GenieClass("Past order or a shopping cart")
class Order extends DataClass {
    @GenieKey()
    public orderId: string;
    @GenieProperty("Items in the order")
    public orderItems: FoodItem[];
    constructor({orderId, orderItems}: {orderId: string, orderItems: FoodItem[]}) {
        super({orderId, orderItems});
        this.orderId = orderId;
        this.orderItems = orderItems;
    }
    @GenieFunction()
    static All(): Order[] {
        return fetchOrdersFromServer();
    }
    @GenieFunction("Create a new order")
    static CreateOrder(): Order {
        return new Order({orderId: randomId(), orderItems: []});
    }
    @GenieFunction("Add an item to the order")
    addItem({foodItem}: {foodItem: FoodItem}) {
        this.orderItems.push(foodItem);
        updateServer();
    }
}
```
Define Object-Oriented States

- Taco Bell
- Taco 3/3
- Mr Sun 3/3
- Crunchwrap
- Quesadilla
- Taro boba

Define UI Components

- Restaurant
- Order
- Item View
- Food Item
- Food Thumbnail

Developer-Coded GUI

Recap: ReactGenie Execute User’s Request within UI Context
Recap: ReactGenie Execute User’s Request within UI Context

Defined States

Order.GetActiveCart().addItems(items:Order.OrderHistory().matching(field:.restaurant,value:Restaurant.current())[0].items)

ReactGenieDSL

Generated Multimodal UI
Lecture Goals

• Why do we need multimodal interactions?

• Introduction to existing multimodal interactions?

• Three problems for multimodal app development

• Introduction to ReactGenie: a multimodal app development framework

• Evaluation of ReactGenie
How good is ReactGenie as a framework?

• For developers:
  • D-RQ1: Assessing the expressiveness of ReactGenie
  • D-RQ2: Cost comparison for expert developers using ReactGenie vs. existing
  • D-RQ3: Ease of learning and usability for novice developers

• For users:
  • U-RQ1: Parser performance with natural language commands
  • U-RQ2: Usability and efficiency of multimodal UIs generated by ReactGenie
We built three apps to demonstrate expressiveness (F-RQ1)

<table>
<thead>
<tr>
<th>App Name</th>
<th>FoodOrdering</th>
<th>Social</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>Order, FoodItem, Restaurant, OrderItem</td>
<td>Post, User, Message</td>
<td>User, Document, SignatureRequest, EmailAddress</td>
</tr>
<tr>
<td>GenieComponent</td>
<td>11</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>OtherComponents</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>GenieFunction</td>
<td>22</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>GenieProperty</td>
<td>18</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>State Code (lines)</td>
<td>835</td>
<td>449</td>
<td>421</td>
</tr>
<tr>
<td>Component Code (lines)</td>
<td>1854</td>
<td>585</td>
<td>446</td>
</tr>
<tr>
<td>Examples (count)</td>
<td>11</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

• Three apps and a wide range of multimodal commands demonstrate the expressiveness of ReactGenie.
• Different UI naturally decomposed to components representing different ReactGenie state classes.
• Only 5% of the code (annotations) must be written to handle multimodal interactions.

Add three of this to my cart
Show me posts from Mark that I have liked before.
Only show me request through this email
A small ReactGenie demo app that can be built within a lab study: ReactGenie Timer

- Classes: Timer, TimeDelta
- Functions: CreateTimer, StartTimer, PauseTimer, ClearTimer, ...
- User can:
  - Create, start, and pause timer with voice.
  - Start/stop timer of a certain category.
  - Filter timer by remaining time.
  - ...
We asked an expert developer to build an App in ReactGenie and GPT FC (F-RQ2)

<table>
<thead>
<tr>
<th>Metric</th>
<th>ReactGenie</th>
<th>GPT-3 Function Calling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to Develop (minutes)</td>
<td>45</td>
<td>177</td>
</tr>
<tr>
<td>Additional Lines of Code</td>
<td>159</td>
<td>523</td>
</tr>
<tr>
<td>Features Supported</td>
<td>Touch, Complex Commands, Navigation</td>
<td>Limited Support</td>
</tr>
</tbody>
</table>

- ReactGenie demonstrates significant advantages in terms of time efficiency and code complexity.
- ReactGenie offers comprehensive multimodal support, enhancing user convenience.
- GPT Function Calling requires extensive code and lacks essential features like UI mapping.
- ReactGenie is a powerful tool for multimodal app development, streamlining the process and improving user experiences.
Novice Developer Study on ReactGenie Usability (F-RQ3)

- Conducted an IRB-approved user study with novice developers
- Developers tasked with building multimodal applications using ReactGenie
- Study Design:
  - Two-part process: Learning phase with tutorial and independent app construction
    - Provided boilerplate code for focus on multimodal features
  - High comprehension of framework functionality
  - Every developer finished the app in just a bit more than one hour!
  - Positive feedback on ease of use and willingness to use ReactGenie in real-life applications

### Metric Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion Time</td>
<td>Average: 109.67 minutes (std = 24.19 minutes)</td>
</tr>
<tr>
<td>Learning Phase</td>
<td>Average: 42.33 minutes (std = 14.90 minutes)</td>
</tr>
<tr>
<td>Independent Building Phase</td>
<td>Average: 67.33 minutes (std = 17.52 minutes)</td>
</tr>
<tr>
<td>Comprehension Quiz</td>
<td>11/12 participants scored 7/7, 1 participant scored 6/7</td>
</tr>
<tr>
<td>SUS Usability Score</td>
<td>Average: 6.08/7 (Ease of Use)</td>
</tr>
<tr>
<td>NASA-TLX Cognitive Load</td>
<td>Average: 21.99/100 (Lower indicates better)</td>
</tr>
<tr>
<td>Willingness to Use</td>
<td>All developers indicated affirmation to ReactGenie; 6/12 participants asked us about using ReactGenie in real-life applications</td>
</tr>
</tbody>
</table>
Evaluate parser’s effectiveness with commands from crowd workers (U-RQ1)

- Used Cloudlicit method for real-world scenario commands
- Participants provided with app screenshots and videos to prompt commands
- Results:
  - 288 commands analyzed after filtering unclear responses
  - Categories of Commands:
    - Simple UI interaction: 100 commands
    - Targeted interactions (complex tasks): 172 commands
  - Out of scope for ReactGenie: 16 commands
  - Parser Accuracy:
    - 101 supported commands: parsed correctly **91%**
    - 71 unsupported commands: generated sensible ReactGenieDSL **53%**
User Experience with ReactGenie-Generated UIs (U-RQ2)

Compare user performance and experience using multimodal UIs vs. GUI-only

• Study Design:
  • Within-subject design with 16 participants
  • Tasks performed on both multimodal UI and GUI-only versions of ReactGenie Food Ordering app

• Measured task completion time, cognitive load (NASA-TLX), and usability (SUS)
  • Cognitive Load: Significantly lower with multimodal UI (NASA-TLX score: 24.6 vs. 34.5)
  • Usability: Higher with multimodal UI (SUS score: 73.0 vs. 63.3)

• Participant Preferences:
  • 11/16 preferred multimodal UI
  • Reasons for preference: Ease of use, efficiency, and clarity in task execution
  • Reason for alternative choice: Poor voice recognition
Recap

• Multimodal interaction history is long, but adoption is limited due to implementation costs.
• Multimodal interactions entail human-computer communication via multiple input/output modes.
• Compared to voice interfaces, multimodal ones are flexible, efficient, clearer, and less error-prone.
• ReactGenie aims to foster multimodal interaction adoption.
  • Merges modern app features and multimodal interface flexibility, ensuring easy development.
  • Utilizes object-oriented state abstraction and declarative UI for modality synchronization.
  • Employs LLMs to expose the app’s entire state, rather than limiting it to individual APIs for voice interfaces.
I hope that you will...

• Build with it 😎

• Break it 😈

• Fix it! 😃