A APPENDIX

A.1 DETAILS OF THE BREWALGORITHM

We provide pseudocode for the core components of BREW, aligning with the stages introduced in Section 3. Each algorithm plays a distinct role in constructing, organizing, or refining the knowledge base over iterative interactions. GENERATEINSIGHTS (Alg. 2) produces concept-aligned insights from annotated rollouts using `ReflAgent`. DEDUPLICATECONCEPTS (Alg. 3) clusters semantically overlapping concepts into a compact meta-concept set. INTEGAGENT incrementally builds and updates per-concept documents using newly generated insights. Finally, EXPANDNODE (Alg. 4) performs MCTS-guided expansions to explore improved document variants, while EVALUATE (Alg. 5) scores candidate KB states using correctness and retrieval-based rewards.

We specify the `IntegAgent` prompt below:

BREW Integrator Prompt

```
# Enhanced Documentation Editor Prompt

You are a meticulous documentation-level editor specializing in
    comprehensive technical reference materials. You will be given a
    list of topic nodes, each containing structured information that
    must be preserved and enhanced with maximum detail retention.

## Input Structure Analysis
Each node contains:
- **Title**: The primary topic identifier
- **Context**: Background information and conceptual foundation
- **How to Use**: Step-by-step instructions, commands, flags,
    parameters, and implementation details
- **When to Use**: Specific scenarios, conditions, and decision
    criteria
- **Best Practices**: Expert recommendations, optimization techniques,
     and common pitfalls to avoid

## Detailed Processing Requirements

### 1. Information Preservation (Zero Loss Policy)
- **Preserve every technical detail**: All command-line flags,
    parameter values, configuration options, file paths, URLs, version
     numbers, and exact syntax
- **Maintain all examples**: Keep every code snippet, sample input/
    output, file names, directory structures, and command sequences
    exactly as provided
- **Retain contextual nuances**: Preserve qualifying language like "
    typically," "usually," "in most cases," "when available," and
    conditional statements
- **Keep quantitative data**: Preserve all numbers, measurements,
    timeframes, limits, thresholds, and statistical information
- **Maintain cross-references**: Keep all mentions of related tools,
    dependencies, prerequisites, and interconnected concepts

### 2. Enhanced Detail Extraction
- **Expand abbreviations**: When encountering shortened forms, expand
    them naturally while preserving the original
- **Surface implicit knowledge**: Make obvious assumptions explicit (e
    .g., "this requires root permissions," "assumes default
    configuration")
- **Clarify relationships**: Explicitly describe how different
    components, options, or steps relate to each other
- **Highlight edge cases**: Emphasize special conditions, exceptions,
    or unusual scenarios mentioned in the source
- **Elaborate on consequences**: When the source mentions outcomes,
    expand on both success and failure scenarios
```

```
### 3. Prose Transformation Guidelines
- **Bullet integration**: Transform each bullet point into 1-3
    complete sentences that naturally flow together
- **Technical precision**: Use precise technical vocabulary while
    maintaining readability
- **Logical flow**: Organize information within each section to follow
     a logical sequence (setup →execution →verification)
- **Contextual embedding**: Weave code snippets and technical terms
    seamlessly into narrative sentences
- **Comprehensive coverage**: Ensure every sub-bullet, nested item,
    and parenthetical note becomes part of the prose

### 4. Structural Requirements
- **Heading hierarchy**: Use `# Title` for each node's main heading
- **Section order**: Maintain Context →How to Use →When to Use →Best
     Practices sequence
- **Paragraph organization**: Create substantial paragraphs (3-6
    sentences) rather than brief statements
- **Transition quality**: Craft smooth bridges between sections and
    between different nodes
- **Code formatting**: Preserve all inline code with backticks and
    maintain proper formatting for code blocks

### 5. Quality Assurance Checklist
Before finalizing, verify:
- [ ] Every piece of source information appears in the output
- [ ] All technical specifications, parameters, and examples are
    intact
- [ ] Code snippets maintain their exact syntax and formatting
- [ ] Prose flows naturally without choppy or fragmented sentences
- [ ] Each section provides comprehensive coverage of its topic area
- [ ] Cross-references and dependencies are clearly explained
- [ ] No section labels or formatting artifacts remain in the prose

## Output Specifications
Generate a single, cohesive markdown document that reads as
    authoritative technical documentation. The result should be
    comprehensive enough that a reader could successfully implement
    the described tools or techniques using only the information
    provided, without referring back to the original nodes.

---

**Input Nodes:**
<NODES>
{node_list}
</NODES>

---

Now, produce the aggregated markdown reference sheet with maximum
    detail preservation and enhanced clarity.
```

Algorithm 2 GenerateInsights: Extract behavioral insights from trajectories

Require: Queries $\mathcal{Q}$, KB $\mathcal{D}$, rubrics
Ensure: Concept-insight pairs $\mathcal{B}$

$\mathcal{B} \leftarrow \varnothing$
for each query $q \in \mathcal{Q}$ **do**
  $\tau \leftarrow \text{LLM}(q, \mathcal{D})$ ▷ Generate trajectory
  label $\leftarrow$ GRADE(τ) ▷ Success/failure
  $(c, i) \leftarrow$ REFLAGENT(τ, rubrics, label)
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(c, i, q)\}$ ▷ Store with source query
end for
return $\mathcal{B}$

Algorithm 3 DeduplicateConcepts: Cluster similar concepts and map queries

Require: Concept-insight-query triples $\mathcal{B}$
Ensure: Meta-concepts $\mathcal{K}$ with mapped queries and insights

Extract all concepts from $\mathcal{B}$
Embed and cluster concepts by similarity
$\mathcal{K} \leftarrow$ cluster representatives
for each $k \in \mathcal{K}$ **do**
  $\mathcal{Q}_k^{\text{train}} \leftarrow$ {training queries that contributed insights to k}
  $\mathcal{Q}_k^{\text{eval}} \leftarrow$ {held-out queries relevant to k}
  $\mathcal{I}_k \leftarrow$ {all insights mapped to concept k}
end for
return $\mathcal{K}$ with associated queries and insights

Algorithm 4 ExpandNode: Generate and evaluate new document variants

Require: Node s, concept k, candidates h, current KB $\mathcal{D}_{\text{current}}$, best docs $\mathcal{D}_{\text{best}}$, tree
Ensure: Updated tree with new evaluated nodes

▷ Generate new insights from concept-relevant queries
$\mathcal{B}_{\text{new}} \leftarrow \varnothing$
for query $q \in \mathcal{Q}_k^{\text{train}}$ **do**
  $\tau \leftarrow \text{LLM}(q, \mathcal{D}_{\text{current}})$
  $(c, i) \leftarrow$ ANNOTATE(τ, rubrics, $\cdot$)
  if c maps to k **then**
    $\mathcal{B}_{\text{new}} \leftarrow \mathcal{B}_{\text{new}} \cup \{i\}$
  end if
end for
▷ Generate and evaluate candidate documents
for $j = 1$ to h **do**
  $d_{k,j} \leftarrow$ INTEGAGENT($k, \mathcal{I}_k \cup \mathcal{B}_{\text{new}}, d_k^s$)
  ▷ Evaluate using hybrid KB with best docs from other concepts
  $\mathcal{D}_{\text{hybrid}} \leftarrow \{d_{k,j}\} \cup \{d_{k'} \in \mathcal{D}_{\text{best}} : k' \neq k\}$
  $R_{k,j} \leftarrow$ EVALUATE($d_{k,j}, \mathcal{D}_{\text{hybrid}}, \mathcal{Q}_k^{\text{eval}}$)
  ▷ Add to tree and backpropagate
  Add $(d_{k,j}, R_{k,j})$ as child of s in tree
  Backpropagate $R_{k,j}$ from new node to root
end for

Algorithm 5 Evaluate: Score document using held-out queries

Require: Document d_k, hybrid KB $\mathcal{D}_{\text{hybrid}}$, eval queries $\mathcal{Q}_k^{\text{eval}}$
Ensure: Reward score R
1: $R^{\text{corr}} \leftarrow 0$
2: $R^{\text{ret}} \leftarrow 0$
3: **for** each $q \in \mathcal{Q}_k^{\text{eval}}$ **do**
4: $\quad R^{\text{corr}} \leftarrow R^{\text{corr}} + \text{EVAL}(q, \text{agent} \oplus \mathcal{D}_{\text{hybrid}})$
5: $\quad R^{\text{ret}} \leftarrow R^{\text{ret}} + \text{MRR}(d_k, q, \mathcal{D}_{\text{hybrid}})$
6: **end for**
7: $R^{\text{corr}} \leftarrow \frac{R^{\text{corr}}}{|\mathcal{Q}_k^{\text{eval}}|}$
8: $R^{\text{ret}} \leftarrow \frac{R^{\text{ret}}}{|\mathcal{Q}_k^{\text{eval}}|}$
9: **return** $\lambda_{\text{corr}} \cdot R^{\text{corr}} + \lambda_{\text{ret}} \cdot R^{\text{ret}}$

A.2 BREW CONFIGURATIONS

Base LLM Configuration For all BREWalgorithm steps, we use the OpenAI GPT-4.1-2025-04-14 model as the underlying language model. To balance exploration and stability, we set the temperature to 0.7 for the `IntegAgent` component to encourage diversity in sampled completions, while all other calls use a temperature of 0.1 for deterministic behavior. The search process employs an expansion width of $e = 3$, a maximum search depth of $k = 3$, and a maximum of $n = 10$ iterations. Reward signals are weighted equally across correctness and retrieval relevance, with $\lambda_{corr} = \lambda_{ret} = 0.5$.

A.3 BASELINE METHODS

We compare BREWagainst two common reasoning baselines. Step-Back Prompting encourages backward reasoning by guiding the model to work from the final task objective back to the initial actions. In-Context Learning augments the input prompt with successful trajectories from related tasks, enabling the model to benefit from relevant prior examples without additional fine-tuning.

A.4 BENCHMARK SPECIFICATIONS

A.4.1 OSWORLD: COMPUTER-USE AUTOMATION

Dataset Overview OSWorld Xie et al. (2024) comprises 369 real-world computer-use tasks spanning 10 distinct applications. The benchmark is divided into train and test sets, with the distribution of tasks across domains shown in Table 2.

Agent Specifications The UI-Tars-7B variant is a 7B-parameter multimodal transformer fine-tuned for graphical user interface understanding. It operates over an action space of PyAutoGUI commands (e.g., click, type, and key presses). The agent integrates a retrieval module that queries a task-relevant knowledge base using the user-provided description, with the top three retrieved items added to the system prompt. Inputs to the model consist of a screenshot of the active UI paired with the natural language task description.

The GTA1-7B configuration adopts a two-agent architecture, consisting of a planner and a grounding module. The planner (GTA-1-7B) generates the high-level action sequence, while the grounding module (OpenAI O3) verifies and refines each action before execution. Knowledge retrieval is incorporated differently for each component: the planner performs a single retrieval at the start of execution, which is persisted in its prompt, whereas the grounding module performs dynamic retrievals at each verification step.

Evaluation Protocol Evaluation uses 134 task-specific scripts designed for automated verification. Success criteria include file state checks (e.g., validating `.xlsx` or `.docx` outputs), UI element validation to confirm correct interaction, and process completion checks to ensure that the intended automation sequence was executed successfully.

A.4.2 τ^2-BENCH: INTERACTIVE TOOL USAGE

Dataset Overview τ^2-Bench Barres et al. (2025b) extends τ-Bench by introducing bidirectional tool-calling capabilities. The dataset covers multiple service-oriented domains, with domain-level task distributions summarized in Table 3.

Domain Characteristics The benchmark spans several domains with distinct task characteristics. The Telecom domain focuses on connectivity troubleshooting, plan modifications, and service activation workflows. The Retail domain includes order processing, return handling, and inventory queries. The Airline domain emphasizes booking modifications and policy-compliant rescheduling scenarios.

Interaction Settings Two interaction modes are defined. In Easy mode, a human proxy (implemented via GPT-4.1) provides detailed guidance to the agent. The knowledge base is built exclusively from Easy mode trajectories, ensuring high-quality demonstrations for learning. In Hard mode, human intervention is minimized. The knowledge base combines both Easy and Hard trajectories, testing the agent's robustness to underspecified or noisy instructions.

Evaluation Criteria Task success is measured using domain-specific verification procedures. These include database state checks to validate final outcomes, status checks for confirming service or connection state, natural language verification to ensure correct confirmation statements appear in dialogue, and action matching to confirm that all required steps are completed. Each domain uses a tailored subset of these checks (e.g., Telecom relies primarily on status checks).

Domain	Test	Train
Calc	45	2
Chrome	44	2
Writer	21	2
Gimp	24	2
Impress	45	2
Os	22	2
Thunderbird	13	2
Multi-apps	99	2
VLC	15	2
VSCode	21	2
Total	**349**	**20**

Table 2: Test and Train samples across different domains in OSWorld.

Domain	Test	Train
Telecom	105	7
Retail	105	7
Airline	44	6
Total	**254**	**20**

Table 3: Task-wise breakdown for τ^2-Bench with assumed 2-shot training samples per domain.

Domain Characteristics

- **Telecom:** Connectivity issues, plan management, service activation
- **Retail:** Order processing, returns, inventory queries
- **Airline:** Booking modifications, policy-compliant rescheduling

Evaluation Criteria Task success determined by:

- **Database Checks:** Final state verification

- **Status Checks:** Service/connection state validation
- **NL Checks:** Confirmation statements in dialogue
- **Action Matching:** Required action sequence completion

Note: Each domain uses specific check combinations (e.g., Telecom uses only status checks).

A.4.3 SPREADSHEETBENCH: REAL-WORLD SPREADSHEET MANIPULATION

Dataset Overview SpreadsheetBench Ma et al. (2024) consists of 912 instructions collected from four major Excel forums and blogs. Each instruction is paired with spreadsheets reflecting authentic, complex user scenarios, often containing multiple tables and non-standard relational structures. The dataset totals 2,729 test cases, averaging three per instruction. A breakdown of cell-level and sheet-level manipulations is shown in Table 4.

Task Settings The benchmark defines two dimensions of evaluation:

- **Granularity:** Instructions involve either *cell-level* manipulations (specific ranges such as `D2:D6`) or *sheet-level* manipulations (entire tables or multi-sheet updates).
- **Evaluation:** Performance is measured using an Online Judge (OJ)-style protocol. The *soft* setting (IOI-style) awards partial credit when only some test cases are solved, while the *hard* setting (ICPC-style) requires solutions to succeed on all test cases.

Agent Configuration We evaluate
texttto4-mini using a function-calling agent connected to a single Python execution tool. The agent translates natural language instructions into Python code for spreadsheet manipulation (e.g., modifying cells, applying formulas, restructuring tables). After each tool call, all formulas in the spreadsheet are recalculated to ensure consistency before proceeding to the next step. This setup provides a controlled environment to assess reasoning, code generation, and execution robustness across diverse spreadsheet tasks.

Granularity	Instructions	Test Cases
Cell-Level	329	986
Sheet-Level	583	1,743
Total	912	2,729

Table 4: Cell-level vs. sheet-level distribution in SpreadsheetBench.

A.5 KB CONSTRUCTION AND RETRIEVAL DETAILS

Training Data Collection

- **OSWorld:** 20 successful trajectories (2 per application domain) and 10 for evals.
- **τ^2-Bench:** 20 trajectories balanced across domains and difficulty settings and 10 for evals.
- **SpreadsheetBench:** Uniformly sample 30 trajectories for training and 10 for evaluation.

All numbers are reported on the remaining train set.

Retrieval Strategy

- **Query Formation:** For each task we take in the seed Natural Language query as the retrieval query.
- **Retrieval Count:** We take top-3 documents for all the retrieval steps
- **Integration Point:** For SPREADSHEET ENCH and **OSWorld** we insert retrievals in the system prompt augmentation. For τ^2-bench we add perfrom retrieval after each user interaction.

	Baseline	max_width=3, max_depth=3	max_width=3, max_depth=10	max_width=10, max_depth=3
OSworld	44.20	47.56	43.83	49.32

Table 5: OSworld difference in MCTS parameters

B Qualitative Analysis

Exploration on MCTS parameters WE evaluate OSworld on two different MCTS parameters.

- Increased Depth: To increase the depth we keep maximum width of the tree as 3 and depth as 10 with max number of iterations as 25. We observe that the Knowledge base over optimizes on the train set leading to a poorer performance on test set.
- Increased Width: For increased width we reverse the parameters where depth is 3 and maximum width is 10 with max iterations 25. We observe many different styles of KBs are generated storing very similar information, these different styles lead to a varied performance on both eval and test set notifying the importance of state search.

We report the numbers on table **??**

C Exemplar Knowledge Bases

C.1 Knowledge base learned for OSWorld

We showcase a small part of knowledge base learned thought BREW . This demonstrate 3 major parts on which each document is aggregated. These parts discuss when to use a piece of information, why to use the information, how to use the information/tool.

```
## Search and Open Files

**When to use**: Locating documents, spreadsheets, images, or
    downloads for editing, conversion, or attachment.

### How to Perform
- Open **File Manager (Nautilus)** from launcher or system dock
- Press `Ctrl + F` or click the search icon
- Enter part of filename, full name, or wildcard (`*.pdf`, `report*`)
- Use right-click →**Open With** to choose the desired application
- Use the sidebar to navigate to **Downloads**, **Documents**, or
    custom folders

### Additional Actions
- Right-click →**Properties** to check modification date or file type
- Sort results by Date, Type, or Name from the top-right dropdown
- Use `F2` to rename files inline

### Example
- Task: "Edit the file titled `sales_report_march.ods`"
  - Search for `sales` in File Manager
  - Confirm `.ods` type and open with LibreOffice Calc

...

## Insert Images

**When to use**: Adding visual elements to documents, presentations,
    emails, or templates.

### How to Perform
- Navigate to **Insert →Image →From File** (in Writer, Impress,
    Thunderbird)
```

```
- Select an image file (`.png`, `.jpg`, `.svg`) from the file dialog
- Use drag handles to resize; right-click →**Wrap** or **Alignment**
    for layout

### Additional Actions
- In GIMP: **File →Open as Layers** to insert image as a new layer
- Use drag-and-drop from file manager into open document windows
- Use **Format →Image** to apply borders, shadows, or color
    corrections (in Writer/Impress)

### Example
- Task: "Insert the logo.png image into the title slide"
  - Open `.odp` file in Impress →Go to Slide 1 →Insert →Image →
      Select `logo.png`

...

## Export as PDF

**When to use**: Required submission format

### How to Perform
- Go to **File →Export As PDF**
- Choose output folder (usually **Documents** or **Downloads**)
- Click **Save**, then confirm the exported file opens correctly

### Additional Actions
- In GIMP or Impress: choose **File →Export As**, then select `.pdf`
    from format list
- Use **Save As** to preserve both editable and exported versions
    separately

### Example
- Task: "Export the flyer.xcf as a PDF"
  - Open in GIMP →File →Export As →Rename to `flyer.pdf` →Click
      Export
```

C.2 BREW KNOWLEDGE BASE FOR τ^2-BENCH

BREW enable use to learn relevant information for tau bench for across the domains in a single knowledge base. This knowledge base is helpful to use relevant actions from the action pool.

```
### Additional Actions

* Inform the user:
  - Refunds via gift card = immediate.
  - Refunds via other methods = -57 business days.

### Example

* Task: "Cancel a T-shirt order placed yesterday"
  * Validate: Status is `pending`
  * Reason: "no longer needed"
  * Confirm
  * Execute tool call


# Exchange Delivered Order

**When to use**:
User wants to swap delivered items for a different variant (e.g., size
    or color).
```

````
**Why to use it**:
To fix sizing or option errors without needing a new purchase.

### How to Perform
- Authenticate user
- Confirm order status is `delivered`
- Get full list of exchange items
> "Please ensure all items for exchange are listed. This step 'cant be
    repeated."
- Ask for refund/payment method
- Confirm:
 > "'Youre exchanging item X for same product, different option.
    Proceed?"
- On confirmation:
 ```python
 request_exchange(order_id="45678", item_exchanges=[...],
    payment_method="paypal")
 ```

### Additional Actions

* Mention: An email will be sent with return instructions
* Validate that the new variant is from the same product

### Example

* Task: "Exchange red shirt for blue in Order #45678"
 * Confirm all exchange items
 * Confirm payment method for difference
 * Execute tool call

### Example

* Task: "Show me my last 2 orders"
 * Authenticate
 * Retrieve and present info

# Deny Unsupported Request

**When to use**:
User asks for an unsupported action (e.g., cancel processed order,
   exchange to different product type, help another user).

**Why to use it**:
To stay compliant with platform policy.

### How to Perform
- Politely reject:
 > "'Im sorry, but I 'cant process that request. 'Its outside the
    allowed scope."

### Example

* Task: "Cancel a processed order"
 * Respond with denial message
# Transfer to Human Agent

**When to use**:
User needs help outside the 'assistants permitted capabilities.

**Why to use it**:
To ensure user gets the right help from trained staff.

### How to Perform
````

```
- Make tool call:
  ```python
  transfer_to_human_agents()
  ```
- Then inform user:
  > "YOU ARE BEING TRANSFERRED TO A HUMAN AGENT. PLEASE HOLD ON."

Example

* Task: "Delete a task"
  * Deny deletion
  * Transfer to human
```

C.3 BREW KNOWLEDGE BASE FOR SPREADSHEETBENCH

```
   Header Extraction
1. Detecting Header Rows
Overview:
To accurately identify header rows, scan the initial region of your
    dataset. This process is crucial for mapping column information
    for further processing.

Approaches:
- Heuristic Checks:
- Look for rows where all cells are strings (e.g., "Name", "Date", "
    Region", "Amount").
- Identify rows with distinctive formatting such as bold text or
    background color.
- Example:
| Name | Date | Region | Amount | |--
    -----|-----------|-----------|--------| | John | 2024-01-01| North
     | 100 |
- Pattern Recognition:
- Use regex to match typical header patterns, such as column names
    starting with uppercase letters.
- Score candidate rows based on the likelihood of being headers.
- Multi-Table Sheets:
- Detect gaps, empty rows, or separators indicating a new table.
- Assign a Table ID to each detected table for later reference.

Edge Cases:
- Merge multi-row headers (e.g., "Sales" over "2024", "2025" becomes "
    Sales 2024", "Sales 2025").
- Fill in missing headers by inferring from context.

2. Assigning and Validating Headers
Overview:
Once headers are detected, assign them programmatically and ensure
    they match expected schema and data types.

Implementation:
- Column Naming:
- Set names in code, e.g., df.columns = ["Name", "Date", "Region", "
    Amount"].
- Schema Mapping:
- Map headers to a standardized schema, using external files or user
    prompts.
- Example:
- Raw header: "Amt"; Mapped header: "Amount"
- Quality Checks:
- Detect duplicate or empty headers ("Date", "Date" becomes "Date_1",
    "Date_2").
- Validate each column's expected data type.
```

```
3. Automation and Usability Enhancements
Overview:
Enhance usability and automation to streamline header extraction and
    user interaction.

Features:
- Freeze Panes:
- Automatically freeze header rows in Excel for easier navigation.
- Highlighting:
- Use colored formatting to visually distinguish headers.
- Example:
- Yellow fill for header row.
- Documentation:
- Log extraction logic and confidence scores for each detected header.
- Integration:
- Build header extraction into ETL pipelines and record process
    metadata.

Block Detection
1. Identifying Block Boundaries
Overview:
Block detection segments data into logical units or tables.

Methods:
- Boundary Detection:
- Find empty rows, repeated labels, or formatting changes.
- Example:
| Name | Amount | |------|--------| | John | 100 | | | | <-- Empty row
     indicates new block | Name | Amount | | Alice| 200 |
- Machine Learning:
- Train classifiers to detect block boundaries based on cell patterns.

Advanced:
- Detect nested blocks or hierarchies using indentation or merged
    cells.
- Identify summary blocks with keywords like "Total" or "Summary".

2. Processing and Tracking Blocks
Overview:
Once blocks are detected, assign IDs and enable block-level analysis.

Actions:
- Block ID:
- Assign unique IDs (e.g., Block_001, Block_002).
- Analysis:
- Perform group-by or aggregation within each block.
- Example:
- Sum "Amount" for Block_001: 100 + 150 = 250

3. Additional Block Actions
Overview:
Enable modular analysis and reporting at the block level.

Features:
- Summary Rows:
- Add computed totals/averages for each block.
- Export/Save:
- Save blocks as separate files or sheets.
- Example:
- Export Block_001 to "block1.csv"

Search for Values or Patterns
1. Search Execution Methods
Overview:
Efficiently locate specific values or patterns in your data.
```

```
Techniques:
- Manual Tools:
- Use Ctrl + F in Excel for quick lookups.
- Programmatic Search:
- Scan all cells using loops or vectorized code.
- Example:
- Find all instances of "North" in the "Region" column.
- Pattern Matching:
- Support exact, wildcard (*Total*), and regex (\d{4}-\d{2}-\d{2} for
    dates).

2. Recording and Highlighting Results
Overview:
Log and visualize search matches for user review.

Actions:
- Logging:
- Record coordinates (e.g., Sheet1, Row 3, Col "Region").
- Highlighting:
- Apply conditional formatting to search hits.

3. Advanced Search Scenarios
Overview:
Handle complex or large-scale search requirements.

Scenarios:
- Merged Cells:
- Search within merged cells or across multiple sheets.
- Export:
- Export found results for further analysis.
- Example:
- Export all rows containing "John" to "john_results.csv"

Writeback Results
1. Output Placement
Overview:
Choose where and how to insert results.

Options:
- Target Columns:
- Select existing or blank columns for output.
- Appending:
- Add new columns for flags, counts, or statuses.
- Example:
- Add "Approved_Flag" column next to "Status".

2. Writing and Styling Results
Overview:
Automate and style the output for visibility.

Methods:
- Formulas/Code:
- Use code (e.g., ws.cell(row, col).value = result) to insert results.
- Styling:
- Bold, borders, or colors for output cells.
- Example:
- Green fill for "Success", red for "Error".

3. Audit and Protection
Overview:
Maintain the integrity and traceability of results.

Measures:
- Lock Columns:
```

```
- Prevent edits to output columns.
- Timestamps/User Info:
- Add audit trail for writebacks.
- Example:
- "2024-06-01, User: admin"

Difference in State
1. Sheet Comparison
Overview:
Identify changes between input and output sheets.

Process:
- Load Sheets:
- Read both sheets into memory.
- Compare Cells:
- Detect differences by position and value.

2. Recording and Reporting Differences
Overview:
Log and report all detected changes.

Actions:
- Log Mismatches:
- Record cell coordinates and values.
- Example:
- Cell B3: "North" →"South"
- Export Diff Report:
- List all detected differences for review.

3. Visualization and Automation
Overview:
Make changes visible and automate validation.

Features:
- Highlight Changes:
- Color code changed cells.
- Automate Checks:
- Integrate diff comparisons into test scripts.

Column Selection
1. Selection Criteria
Overview:
Choose relevant columns for analysis.

Methods:
- Labels/Indices:
- Select by name or position.
- Dynamic Rules:
- E.g., all numeric columns.
- Assign Roles:
- Example: "ID", "Date", "Metric"

2. Preparation and Validation
Overview:
Prepare columns for consistent use.

Actions:
- Rename/Relabel:
- Standardize column names.
- Validate Types:
- Ensure columns are of expected type.
- Example:
- "Date" column as datetime.

3. Reusability
```

```
Overview:
Save and reuse column selections.

Features:
- Presets:
- Save selection profiles.
- Downstream Use:
- Use validated columns in subsequent processes.

Filter Rows
1. Filtering Methods
Overview:
Refine your dataset with filters.

Techniques:
- Spreadsheet Tools:
- Use built-in filters.
- Code Logic:
- Filter with code (e.g., df[df['Status'] == 'Approved']).
- Multiple Criteria:
- Combine conditions (AND/OR).
- Example:
- Status = "Approved" AND Amount > 100

2. Helper Columns and Complex Filters
Overview:
Simplify filtering using helper columns.

Actions:
- Helper Columns:
- Compute intermediate flags.
- Document Logic:
- Record filtering rules for audit.

3. Post-Filter Actions
Overview:
Visualize and export filtered data.

Features:
- Highlighting:
- Grey-out filtered-out rows.
- Export:
- Save the filtered dataset.

Merge Tables
1. Key-Based Merging
Overview:
Combine tables using shared keys.

Techniques:
- Join Operations:
- Use VLOOKUP, JOIN, or code merges.
- Example:
- Merge "Customer_ID" from two tables.
- Align Data:
- Match on columns like "ID", "Name".

2. Stack-Based Merging
Overview:
Append tables when keys 'arent needed.

Methods:
- Vertical Append:
- Combine rows from similar tables.
- Deduplicate:
```

```
- Remove duplicate records.

3. Tracking and Audit
Overview:
Track source and unmatched records.

Actions:
- Source Column:
- Add "Source" to indicate origin.
- Highlight Unmatched:
- Mark or export mismatched rows.

Pivot or Unpivot
1. Pivoting Data
Overview:
Summarize data using pivots.

Methods:
- PivotTables:
- Group by row/column dimensions.
- Example:
- Sum "Amount" by "Region".
- Aggregation:
- Choose SUM, AVG, COUNT, etc.

2. Unpivoting (Melting) Data
Overview:
Reshape data from wide to long format.

Techniques:
- Melt Operations:
- Convert columns into rows.
- Example:
-
| Year | Sales_2019 | Sales_2020 | |------|------------|------------|
→
| Year | Sales_Year | Value |
- Flexible Restructuring:
- Selectively unpivot non-ID columns.

3. Post-Pivot Actions
Overview:
Prepare pivoted data for export.

Features:
- Flatten Pivot Table:
- Convert back to flat for further analysis.
- Reorder/Rename:
- Clarify pivoted fields.

Map with Lookup Tables
1. Mapping Techniques
Overview:
Standardize data using lookups.

Methods:
- Functions:
- Use VLOOKUP, merge with dictionaries.
- Code-to-Label:
- Example:
- Code "N" →Label "North"

2. Application and Fallbacks
Overview:
Apply lookups and handle missing values.
```

```
Actions:
- Apply Mappings:
- Across selected columns.
- Handle Missings:
- Use defaults for missing codes.

3. Audit and Display
Overview:
Ensure mapping transparency.

Features:
- Cache Mappings:
- Store for repeated use.
- Display Codes/Labels:
- Show both for clarity.

Fill Missing Data
1. Choosing Fill Methods
Overview:
Impute missing data appropriately.

Techniques:
- Forward/Backward Fill:
- Fill gaps with prior/next value.
- Default Values:
- Use fixed placeholder (e.g., 0, "Unknown").
- Contextual Example:
- Dates: Fill missing month with last known month.

2. Application and Auditing
Overview:
Apply fills and flag for review.

Actions:
- Targeted Filling:
- Apply to specific columns/rows.
- Flag Filled Cells:
- Highlight for later review.

3. Documentation
Overview:
Keep fill logic transparent.

Features:
- Record Logic:
- Document assumptions and methods.
- Audit Trail:
- Track all changes.

Flag Rows or Cells
1. Defining Flag Rules
Overview:
Establish criteria for flagging.

Examples:
- Simple Rule:
- Flag where Amount < 0
- Complex Rule:
- Flag where Status = "Pending" and Amount > 1000

2. Applying Flags
Overview:
Insert flags and summarize.
```

```
Actions:
- Flag Column:
- Add "Flag" column with "Yes"/"No".
- Export Flagged Rows:
- Save for further inspection.

3. Advanced Flagging
Overview:
Use multiple criteria and document.

Features:
- Multi-Criteria:
- Combine several rules for granular checks.
- Notes:
- Document flagging rationale.

Sort Data
1. Setting Sort Criteria
Overview:
Organize data for analysis.

Options:
- Sort Columns:
- By value, ascending/descending.
- Multi-Level:
- E.g., sort by "Region", then by "Amount".

2. Applying Sorts
Overview:
Implement sorting programmatically or manually.

Methods:
- Spreadsheet Tools:
- Built-in sort features.
- Code:
- E.g., df.sort_values(['Region', 'Amount'])

3. Post-Sort Actions
Overview:
Finalize sorted data.

Actions:
- Renumber Rows:
- Update indices.
- Highlight Extremes:
- Mark top/bottom values.

Validate Data
1. Validation Checks
Overview:
Ensure data meets required standards.

Checks:
- Type:
- Ensure numeric columns contain numbers.
- Range:
- E.g., "Amount" > 0.
- Pattern:
- Date columns match YYYY-MM-DD.
- Business Rule Example:
- "Start Date" < "End Date"

2. Marking and Reporting
Overview:
Visualize and report errors.
```

```
Actions:
- Highlight Invalids:
- Color-code errors.
- Export Summary:
- Table of error counts and locations.

3. Integration in Workflow
Overview:
Make validation a routine part of processing.

Features:
- Pre-Processing Step:
- Validate before analysis.
- Automation:
- Integrate into data pipelines.

Split Sheets or Data
1. Defining Split Rules
Overview:
Segment data for modular analysis.

Methods:
- By Category:
- E.g., split by "Region".
- By Date Range:
- E.g., split by year.

2. Exporting Segments
Overview:
Save segments for separate use.

Actions:
- Export Files:
- "North_Region.csv", "South_Region.csv"
- Consistent Formatting:
- Ensure identical columns and styling.

3. Automation and Documentation
Overview:
Automate splitting and track provenance.

Features:
- Automation:
- Use scripts/macros for repeated splits.
- Documentation:
- Record rules and export logs.
```

D QUALITATIVE ANALYSIS OF BREW-GENERATED KNOWLEDGE BASES

This section presents a comprehensive qualitative analysis of knowledge bases generated through the BREW technique applied to two distinct agent training environments: OSWorld and τ^2Bench described in the section before. The analysis examines knowledge representation patterns, procedural sophistication, and domain-specific learning characteristics extracted from CUA agent behaviors, providing insights into the effectiveness and scope of knowledge distillation techniques across diverse task environments.

D.1 Cross-Domain Knowledge Base Analysis

D.1.1 Base Structure & Organization

Schema Consistency and Evolution: Both knowledge bases demonstrate consistent structural schemas, though adapted to their respective domains. The OSWorld KB employs a four-part schema (contextual triggers, procedural steps, extended capabilities, concrete instantiation), while the τ^2Bench KB extends this to a five-part structure, adding explicit purpose rationale ("Why to use it"). This evolution suggests that BREW adapts its extraction patterns to domain-specific requirements—conversational commerce demands explicit justification for actions due to customer interaction contexts.

Taxonomic Organization Principles: The OSWorld KB reveals a **capability-based taxonomy** organized around computational tasks: file operations, document processing, inter-application workflows, and data visualization. Each category represents a distinct computational domain with specific tool requirements and interaction patterns. In contrast, the τ^2Bench KB employs a **lifecycle-based taxonomy** structured around transactional states: order creation, modification, fulfillment, and post-delivery operations. This organizational difference reflects fundamental domain characteristics—desktop automation focuses on tool orchestration, while conversational commerce centers on process management.

Hierarchical Task Decomposition: Both KBs demonstrate sophisticated hierarchical reasoning, but through different decomposition strategies. OSWorld exhibits **technical decomposition**, breaking complex operations like "Create Charts from Data" into constituent technical steps (data selection, chart insertion, customization, formatting). τ^2Bench shows **process decomposition**, structuring operations like order modification into authentication, validation, confirmation, and execution phases. This suggests BREW successfully identifies domain-appropriate decomposition strategies rather than applying uniform patterns.

Knowledge Boundary Definition: Both KBs explicitly encode operational boundaries, but through contrasting mechanisms. OSWorld boundaries are **capability-constrained**—determined by available applications and system resources. τ^2Bench boundaries are **policy-constrained**—explicitly defined through "Deny Unsupported Request" patterns and escalation protocols. This difference highlights how knowledge extraction adapts to domain-specific constraint types.

D.1.2 Procedural Knowledge Grounding

Context-Dependent Action Selection: Both domains demonstrate sophisticated context awareness, but grounded in different environmental factors. OSWorld exhibits **application-context sensitivity**, where identical operations (e.g., image insertion) require different procedures across LibreOffice Writer, Impress, GIMP, and Thunderbird. The agent learned application-specific affordances and interaction patterns rather than generic command sequences. τ^2Bench demonstrates **state-context sensitivity**, where available actions depend on order status (pending vs. delivered), payment methods, and authentication levels. This reveals learned understanding of business process constraints and temporal operation windows.

Error Prevention and Validation Workflows: Both KBs incorporate sophisticated error prevention mechanisms, but grounded in domain-specific failure modes. OSWorld emphasizes **technical validation**: file integrity checks ("confirm the exported file opens correctly"), application state verification, and multi-step confirmation for irreversible operations. τ^2Bench emphasizes **transactional validation**: authentication cascades, confirmation dialogues with standardized templates, and explicit user consent protocols. The emergence of defensive programming practices across both domains suggests these represent fundamental principles of reliable agent behavior.

State-Dependent Decision Logic: The procedural knowledge in both domains demonstrates sophisticated state machine reasoning. OSWorld exhibits **application state awareness**—understanding when applications are ready for input, when files are loaded, and when operations can be safely executed. Window management and application switching reveal learned understanding of desktop metaphors and resource constraints. τ^2Bench demonstrates **business process state awareness**—finite state machine reasoning where order lifecycle states determine available operations. The agent learned that pending orders enable modification while delivered orders unlock return workflows, indicating internalized understanding of business logic constraints.

Security and Authentication Grounding: While OSWorld operates in a trusted desktop environment with minimal explicit security concerns, τ^2Bench reveals pervasive **authentication-first paradigms**. Nearly every transactional operation begins with identity verification through email, name, and zip code combinations. The KB demonstrates **graduated security reasoning**: information retrieval requires basic authentication while financial transactions trigger rigorous verification protocols. This contrast highlights how procedural knowledge adapts to domain-specific security requirements.

Cross-Application vs. Cross-Process Orchestration: OSWorld demonstrates **technical orchestration**—coordinating multiple applications (Chrome, LibreOffice suite, File Manager, GIMP) to accomplish complex workflows. The "Navigate Between Applications" section reveals learned behaviors for window management, application switching, and resource coordination. τ^2Bench exhibits **process orchestration**—coordinating authentication, validation, confirmation, and execution phases across different operational contexts. Both forms of orchestration require sophisticated temporal reasoning and constraint management, but applied to different environmental complexity types.

Failure Mode Internalization: Both KBs reveal learned understanding of domain-specific failure modes. OSWorld incorporates file validation, application crash recovery suggestions, and verification steps for critical operations. τ^2Bench includes explicit escalation protocols ("Transfer to Human Agent"), policy compliance mechanisms, and irreversibility warnings for financial operations. The consistent emergence of failure-aware procedures suggests that agents successfully internalize risk assessment and mitigation strategies during training.

Domain-Specific Communication Patterns: The procedural knowledge reveals distinct communication paradigms appropriate to each domain. OSWorld procedures are **task-oriented** with minimal user interaction—focusing on efficient command execution and verification. τ^2Bench procedures are **dialogue-oriented** with standardized customer interaction templates, confirmation protocols, and expectation management communications. This adaptation demonstrates that BREW extracts not just procedural logic but domain-appropriate interaction modalities.

The cross-domain analysis reveals that BREW successfully extracts procedural knowledge that is both **structurally consistent** (following learnable organizational patterns) and **contextually grounded** (adapted to domain-specific constraints, failure modes, and interaction requirements). This dual capability suggests significant potential for knowledge transfer across related domains while maintaining appropriate domain-specific adaptations.