CRAFT: AN AGENTIC SPREADSHEET FORM FILLING SYSTEM WITH TEMPLATE AWARENESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Spreadsheet form filling requires agents to consolidate external evidence, ground values to precise cells, and preserve irregular template structure. Errors in early edits can overwrite labels or misalign fields, undermining later decisions. We propose `CRAFT`, a template-aware agent framework that connects reflective validation to constrained local repair. Instead of treating reflection as a free-form request to regenerate the workbook, `CRAFT` grounds detected errors to spreadsheet regions, restores corrupted template state when necessary, and re-grounds plausible writable slots before subsequent edits. A Rectangle-Aware Slot Grounder (RASG) proposes writable cells, while label–slot hints and protected regions constrain subsequent edits. We introduce `FormFillBench`, with 327 forms across Instruction-Only and Multi-File tracks. Compared with the strongest baselines, `CRAFT` improves pair accuracy by 8.51 and 23.38 percentage points on these tracks, respectively. Component-removal experiments support structural adjudication and slot re-grounding within the pipeline, and the framework retains its relative advantage among the methods evaluated with a second backbone.

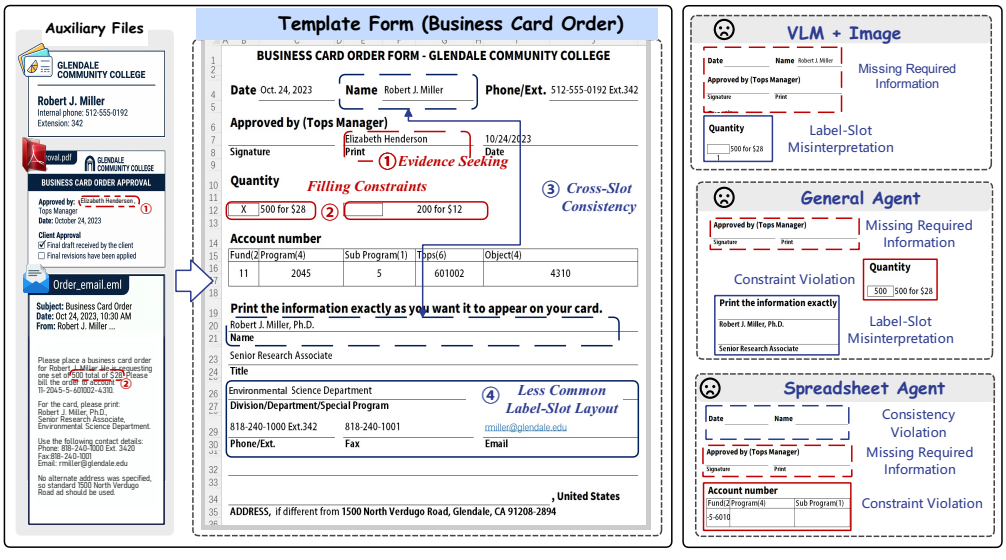

Figure 1: **Example of spreadsheet form filling and its key challenges.** The agent needs to gather evidence from auxiliary files, satisfy field-level constraints, preserve cross-field consistency, and interpret diverse form layouts.

1 INTRODUCTION

Spreadsheet form filling requires an agent to recover information from multiple external sources and write it into a structured spreadsheet artifact without damaging its template. Unlike conventional tabular spreadsheets with regular row–column structures, forms such as reimbursement sheets and compliance questionnaires (University of California, Berkeley, 2026; Clark County, Washington,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2025) organize information through label–slot relationships and spatial layout, where labels and writable regions may be separated, repeated, or span merged cells. Recovering the correct value is therefore insufficient: the agent must also ground it to the correct location while preserving the surrounding structure.

This challenge becomes more pronounced when the required information is scattered across heterogeneous sources. Consider a business-card order supported by an approval PDF, an email, and an old business card (Figure 1). The same person appears in multiple sections, while an account number must be split across designated subfields. An agent can retrieve the right value yet write it into the wrong occurrence of a repeated label, combine subfields into one cell, or overwrite the label itself. Once such an edit changes the template, later decisions can inherit the error. Repair must therefore recover the local template context as well as the intended value.

Existing work addresses important parts of this problem. Spreadsheet agents support reasoning and manipulation (Li et al., 2023; Chen et al., 2025; Wang et al., 2026; Zhu et al., 2025), while visual form agents study interaction with web forms or image-based documents (Li et al., 2025a; Toles et al., 2026). General agent frameworks coordinate tools and heterogeneous inputs (LangChain, Inc., 2026; OpenAI, 2026c; CrewAI, 2026). However, reliable spreadsheet completion requires more than recovering the correct values. The agent must also preserve the template, associate each label with the correct writable region, and maintain consistency across different parts of the form. Our pilot study finds that simply changing the relative position of a label and its corresponding slot can substantially affect the performance of existing systems, even when the information to be filled remains unchanged.

We propose `CRAFT`, a **C**onstrained and **R**eflective **A**gent-based system for spreadsheet **F**orm filling with **T**emplate awareness. Its key idea is to couple error diagnosis with restoration and re-grounding: a suspicious region carries the evidence needed to decide what to restore, where to refill, and which cells to protect. `CRAFT` first produces an initial filled workbook from the instruction, source evidence, and aligned structural and visual views of the template. It then converts validation signals into localized repair state: suspicious regions are identified, template corruption is restored when necessary, and plausible writable slots are re-grounded before any corrective edits are allowed. RASG provides candidate writable cells for these regions, while reliable and unaffected cells are carried forward as explicit protection constraints.

Our contributions are threefold. First, we develop a template-aware validation-and-repair framework that connects evidence consolidation, error localization, and constrained spreadsheet edits. Second, we introduce RASG, which uses rectangular slot geometry and boundary predictions to guide local repair under irregular layouts. Third, we construct `FormFillBench`, comprising 327 forms across Instruction-Only and Multi-File tracks, and evaluate `CRAFT` against 11 baselines. `CRAFT` improves pair accuracy over the strongest baseline by 8.51 and 23.38 percentage points on the two tracks, respectively. Component-removal experiments assess structural adjudication, slot re-grounding, and planning within the full pipeline, and an additional backbone evaluation tests the framework beyond the default model.

2 Task and Motivation

Spreadsheet form filling. Given a template form $\mathcal{F}$, an instruction $\mathcal{X}$, and auxiliary files $\mathcal{S}$, the task is to produce filled entries $\mathcal{O} = \{(\ell_i, v_i, R_i)\}_{i=1}^{N}$. Here ℓ_i identifies a label, v_i is its value, and R_i is the target cell or range. A label identifier combines its text with an occurrence index to distinguish repeated labels. Unsupported values are assigned null. Applying these entries should fill the intended slots while preserving static labels, formatting, merged regions, and other template constraints. Appendix A gives the full form representation and examples.

Predicted and ground-truth entries are aligned using their composite label identifiers. For each aligned pair, correctness requires both value agreement and exact spatial grounding:

$$\mathrm{Correct}(e_i, e_i^*) = \mathbb{I}[\mathrm{MatchValue}(v_i, v_i^*) \wedge (R_i = R_i^*)]\,. \tag{1}$$

Value matching accepts semantically equivalent strings, such as alternative date formats. Values spanning multiple cells or lines are concatenated before comparison. This separates semantic equivalence from the stricter requirement that values occupy the correct spreadsheet range.

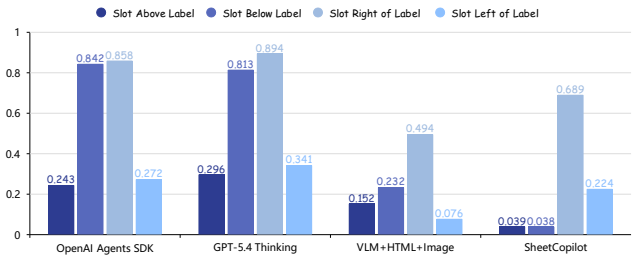

Figure 2: Pilot Study of Spreadsheet Form Filling. Samples are from two common categories of real spreadsheet forms (i.e., mixed label-slot layout and repeated-section forms).

Why localized repair? We conduct a pilot study to investigate how sensitive existing systems are to variations in label–slot layouts. We randomly select 15 mixed-layout and repeated-section forms and expand them into 57 pilot instances by varying label–slot positions while keeping filling semantics fixed. For this diagnostic study, we additionally include GPT-5.4 Thinking (OpenAI, 2026a) to examine layout sensitivity under a reasoning-oriented model, complementary to the lightweight backbones adopted in the main evaluation for large-scale experimentation under a practical inference budget. Figure 2 shows that performance depends on label–slot direction; the left-label/right-slot arrangement is generally favorable, although sensitivity varies across systems. Failures include writing into labels, selecting neighboring cells on the wrong side, and producing inconsistent entries across repeated sections; error locations also vary across runs. The full study appears in Appendix B. These observations motivate validation against the original template and explicit re-grounding of suspicious regions before further edits.

3 CRAFT

Architecture. Figure 3 illustrates the overall architecture of `CRAFT`, which consists of three modules coordinated by an agent-based planner. ***(1) Form Filling Preprocessing*** normalizes heterogeneous auxiliary files into structured evidence and constructs a dual-view spreadsheet representation with an executable view for precise manipulation and a layout-aligned visual view for VLM-based reflection (Section 3.1). ***(2) Reflective Region Adjudication*** combines global screening and structure-aware local validation to identify reliable and suspicious cells, which are consolidated into suspicious ranges for repair (Section 3.2). ***(3) Progressive Slot Re-grounding and Repair*** restores corrupted template content, re-identifies plausible writable slots, matches them to labels, and performs constrained local refill (Section 3.3).

Workflow. Given a template form, user instruction, and auxiliary files, `CRAFT` first preprocesses the inputs and produces an initial prefilled form. The planner applies reflective adjudication to localize suspicious ranges and progressive repair to restore template structure and refill the affected slots. Execution is budgeted: the evaluated configuration permits at most one reflection round and six patch steps after prefilling (Section 5.1). The returned workbook may therefore retain unresolved errors.

3.1 Form Filling Preprocessing

Evidence for form filling is often scattered across heterogeneous sources, such as emails, PDFs, and spreadsheets, whose structural and spatial cues can be lost when flattened into one-dimensional sequences (Xu et al., 2020; Li et al., 2025b). Our preprocessing converts these sources into a unified, structure-aware evidence representation for downstream form filling.

Dual-View Spreadsheet Representation. `CRAFT` first parses the spreadsheet template to recover cell coordinates, and determines the exact form range from the maximal populated row–column extent. Then, `CRAFT` derives two complementary views over this range. The *executable view* retains cell coordinates, content, formatting, merged regions, writable and non-writable areas, and constraints such as dropdown candidate values. The *layout view* renders structured markup into a

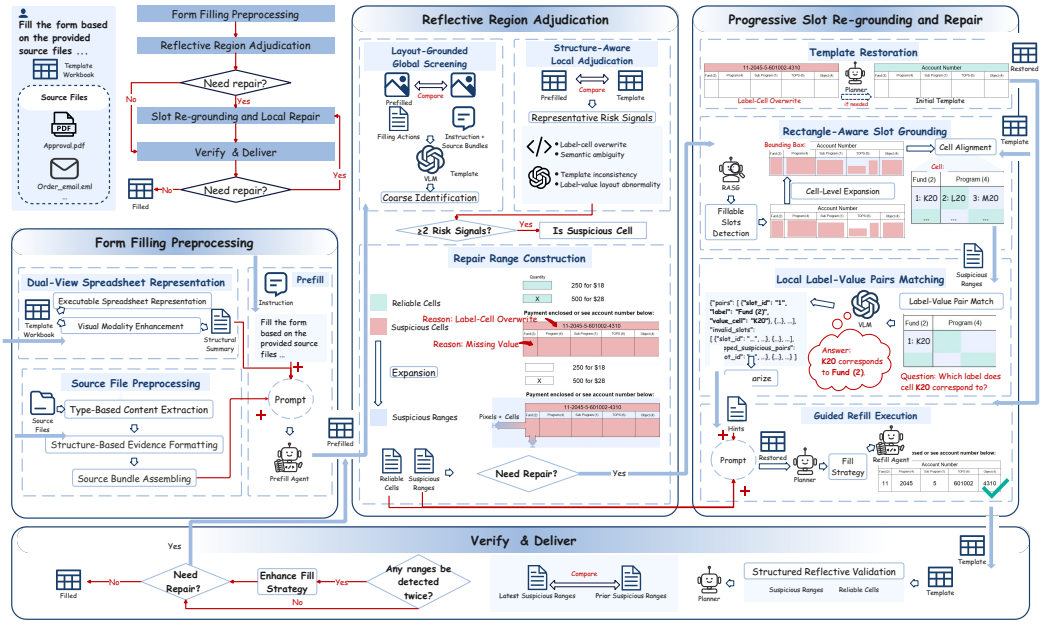

Figure 3: CRAFT combines evidence-based prefilling, reflective region adjudication, and constrained local repair. The planner carries validation evidence and protected cells between stages.

sheet image and records its alignment with spreadsheet coordinates. The first view supports precise edits and programmatic checks; the second exposes spatial relationships for VLM-based validation. The alignment between the two views bridges visual understanding and spreadsheet manipulation. It allows the system to ground VLM-identified anomalies to concrete cells or ranges for subsequent repair.

Source File Preprocessing. CRAFT performs *Type-Based Content Extraction*, *Structure-Based Evidence Formatting*, and *Source Bundle Assembling*. Extraction preserves source-specific cues: text spans and sections, structured label hierarchies, table headers and records, email headers and body sections, and PDF text blocks, tables, and layout anchors. Lightweight signals—label–value regularity, section patterns, and repeated records—then guide formatting into label–value pairs, sectioned text, tables, hierarchical objects, or hybrid views. Weakly structured sources retain grounded text spans. The resulting source bundle provides structured evidence for the planner.

Planner-guided prefilling. The planner combines the instruction, template summary, and source bundle to associate target regions with relevant evidence and establish filling policies, including how to handle unsupported fields. A prefill agent executes this plan through spreadsheet tools. When the workbook is too large to inline, the agent instead inspects its structure and reads small local ranges relevant to each filling decision. The resulting workbook is an initial candidate for validation, not an assumption of correct completion.

Example 1 (Preprocessing)**.** In the running case (Figure 3), an approval PDF, an email, and a business-card image supply the evidence. Signals such as `kv_line_ratio` and `section_line_ratio` guide formatting into sectioned blocks, including "root" and "Use the following contact details." The planner combines these source bundles with the instruction and aligned template views. It assigns A8:T11 to approval information from Approval.pdf and records that the account number must be split across designated fields.

3.2 Reflective Region Adjudication

The prefilled form may contain plausible but incorrect filling actions, such as overwriting labels, placing values into adjacent cells, or violating form-specific constraints. These errors are difficult to detect in isolation and can substantially degrade downstream accuracy (Stechly et al., 2025; Huang

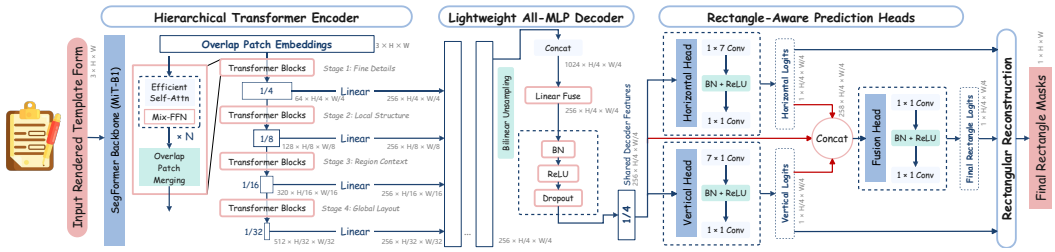

Figure 4: Rectangle-Aware Slot Grounder (RASG). A SegFormer-B1 backbone supports rectangle prediction and auxiliary horizontal and vertical boundary heads.

et al., 2024). To address this issue, CRAFT performs (1) local reflective, evidence-grounded validation to identify reliable and suspicious cells, and (2) range construction to consolidate suspicious cells into compact regions for downstream repair.

Global screening and local adjudication. In the first stage, a VLM compares aligned views of the original and prefilled workbooks, conditioned on the instruction and edit history. It identifies reliable cells and suspicious regions, including missing values, abnormal overwrites, and inconsistent placement. In the second stage, Structure-Aware Local Adjudication (SALA) diagnoses candidates using spreadsheet-specific signals. Programmatic checks detect overwritten static template cells; VLM judgments assess label–value ordering, separation, and other layout-grounding risks. The local stage marks a cell suspicious when at least two risk signals fire, reducing repairs triggered by a single noisy cue. This threshold applies to local adjudication; a coarse warning from global screening can also contribute a repair candidate.

Repair range construction. The system merges the two stages' judgments and expands suspicious cells into coherent local ranges. Expansion first occurs in spreadsheet space, treating merged cells as atomic units, then in pixel space to retain borders and nearby layout context. Each range carries its error types, validation provenance, restoration indicator, and reliable cells. A template-integrity violation triggers restoration before refill, whereas a layout or content anomaly may require only a local rewrite. The planner uses these outputs to decide which ranges enter repair. The representation preserves the context needed for grounding while protecting reliable content outside, and within, the selected region.

Example 2 (Reflective Region Adjudication)**.** For the running case in Figure 3, the Reflective Region Adjudication round produces several suspicious cells (or merged-cell regions). Concretely, `K19:T19` is identified by the Structure-Aware Local Adjudication stage because it triggers both *label-cell overwrite* and *label–value layout abnormality*, while `K21:T22` is identified by the Layout-Grounded Global Screening stage due to an explicit missing value. These suspicious cells are first unionized and expanded in spreadsheet space to form a compact cell-level repair range, `G18:T23`. This cell range is then mapped to its corresponding rendered region in the layout view, where an additional pixel-level margin is added to preserve surrounding layout and semantic context for subsequent repair.

3.3 Progressive Slot Re-grounding and Repair

Template restoration. Within a range flagged for template-integrity violations, CRAFT compares cell values, styles, and merged-range structure against an immutable template snapshot. It restores overwritten template content while preserving cells already judged reliable. This recovers the local structure on which subsequent grounding depends.

Rectangle-aware slot grounding. RASG localizes plausible writable cells in the restored layout view (Figure 4). A SegFormer-B1 backbone (Xie et al., 2021) encodes four scales of features, which are projected, upsampled, and concatenated into a shared decoder representation. Three heads predict rectangle logits and horizontal and vertical boundary cues. Boundary predictions rectify and reconstruct rectangle masks, which are aligned with candidate writable cells using area-based thresh-

olds. These candidates narrow the locations considered by the refill agent. We provide qualitative visualizations of the directional boundary cues and reconstructed slot predictions in Appendix C.4.

RASG is trained in three stages: synthetic spreadsheet forms provide bounding-box and cell-level supervision; CommonForms (Barrow, 2026) supplies more varied PDF layouts with bounding-box supervision; and a manually filtered, annotated FUSE subset (Barik et al., 2015) provides adaptation to real spreadsheet forms with both supervision types. This progression introduces more realistic layouts while retaining supervision for spreadsheet slot localization. If no reliable slot is found, the controller falls back to guarded local repair using the suspicious range, template summary, and protection policy.

Label matching and constrained refill. A VLM checks each candidate value cell and associates valid candidates with nearby labels. Valid matches become positive refill hints; rejected candidates become negative hints excluding non-target locations. Context-only hints retain useful label information when a target cannot yet be selected. The refill agent combines these hints with the instruction, restored workbook state, and validation feedback to generate a local plan specifying allowed edits and protected cells. It then applies spreadsheet actions within this specification. Thus, reflection is connected to explicit spatial targets and preservation constraints rather than a free-form request to regenerate the form.

Example 3 (Progressive Slot Re-grounding and Local Repair)**.** For the same running case in Figure 3, reflective validation identifies the suspicious range `G18:T23`. Since this range triggers the `label-cell overwrite` signal, `CRAFT` first applies template restoration to recover the original template content, including the overwritten label Account Number. The restored local region is then passed to the RASG, which detects a set of plausible writable slots and then aligns them with candidate cells such as `K20`, `L20`, and `M20`. Based on these candidates and the rendered local context, a VLM further performs local label–value matching and determines, for example, that cell `K20` corresponds to the label "`Fund (2)`". These outputs are summarized into structured refill hints, which guide the refill agent to perform constrained local rewriting within the suspicious range and correctly fill the segmented account-number fields.

4 FORMFILLBENCH

Despite rapid progress in LLM-based agents for document interaction and spreadsheet automation (Lin et al., 2025c; Chen et al., 2025), existing form filling benchmarks mainly target web-based or document-centered settings (Li et al., 2025a; Toles et al., 2026), while existing spreadsheet evaluation efforts focus on spreadsheet understanding and manipulation rather than spreadsheet form filling (Rahman et al., 2020). To our knowledge, spreadsheet form filling has not been explicitly studied as a dedicated benchmark task. To address this gap, we introduce `FormFillBench`.

`FormFillBench` contains 327 form instances across two tracks. The *Instruction-Only Track* has 200 forms and 5,499 label–value pairs; all filling information is supplied in a natural-language instruction. The *Multi-File Track* has 127 forms and 2,288 pairs, with roughly five supporting files per case and 11 file types overall. It tests evidence consolidation in addition to spatial grounding.

Templates come from FUSE (Barik et al., 2015), VEUSES and Venron2 (Xu et al., 2017; Dou et al., 2016), public online sources, and synthetic templates generated with Claude Sonnet 4.6 (Anthropic, 2026). For the Instruction-Only Track, Gemini 3.1 Flash-Lite-Preview generates instructions from rendered forms and key annotations. Multi-File cases combine open-source resources and synthetic enterprise documents under a standardized instruction to use explicitly supported information. Candidate samples undergo automated consistency checks and manual verification. Appendix D details construction, file distributions, and five form archetypes spanning regular label–slot layouts, mixed layouts, repeated sections, mixed sections, and bordered blocks.

The two tracks separate complementary difficulties. Instruction-Only cases expose whether a system can ground supplied values into the right locations; Multi-File cases additionally require source selection and evidence consolidation. Structural diversity occurs within both tracks: regular left–right layouts coexist with mixed label–slot directions, repeated groups, and sections with different internal organizations. The archetypes summarize these patterns rather than constitute a strict taxonomy. This makes it possible to inspect where performance varies instead of relying solely on an aggregate accuracy.

Table 1: Primary benchmark results (%). Pair accuracy requires both the value and target range to be correct; E2E requires every pair in a form to be correct. All 11 baselines are shown. Complete metrics appear in Table 5.

	Instruction-Only		Multi-File	
Method	**Pair**	**E2E**	**Pair**	**E2E**
VLM+HTML+Image	51.17	13.5	32.60	12.60
VLM+HTML	49.26	11.0	26.31	7.87
VLM+Excel Skills+Coding	10.66	0.0	0.26	0.0
LangGraph	44.28	10.0	33.65	12.60
OpenAI Agents SDK	45.64	9.5	43.14	16.54
CrewAI	41.77	9.5	40.25	18.90
ReAct	41.66	11.0	34.00	13.39
SheetAgent	8.89	0.0	0.26	0.0
SheetBrain	10.98	0.0	0.26	0.0
SheetCopilot	33.02	7.5	0.26	0.0
SheetMind	10.69	0.0	0.26	0.0
CRAFT	**59.68**	**15.5**	**66.52**	**43.31**

Metrics. *Pair accuracy* measures the proportion of target pairs whose values and ranges are both correct. *End-to-end (E2E) accuracy* measures the fraction of forms with every target pair correct, including fields that should remain blank. These primary metrics apply to both tracks and measure target-entry correctness, rather than a separate audit of every formatting or template-preservation constraint. Supplementary metrics distinguish filling from leaving fields blank: fill precision, recall, and F1 for Instruction-Only, and fill coverage (reported as Fill Accuracy) for Multi-File. Their definitions and complete scores are given in Appendix D and Table 5.

5 Experiments

5.1 Experimental Setup

We compare CRAFT with 11 baselines: three direct VLM variants (HTML, HTML plus image, and coding with basic Excel skills); four general agent methods (LangGraph, OpenAI Agents SDK, CrewAI, and ReAct); and four spreadsheet agents (SheetAgent, SheetBrain, SheetCopilot, and SheetMind). Section 6 discusses these methods. Target spreadsheets are normalized to HTML and non-spreadsheet sources to Markdown. The HTML-plus-image baseline additionally receives a rendered sheet image. Tool-based methods share a spreadsheet interface, while direct-prompting methods operate on their provided renderings.

Unless otherwise specified, CRAFT and agent-based baselines use Gemini 3.1 Flash-Lite-Preview (Google, 2026) with temperature 1.0. The same model acts as the semantic-only judge. CRAFT uses structure-scanning, range-reading, and cell-writing tools, with up to 10 main interaction steps, at most one reflection round, and up to 6 patch steps. Agent-based baselines use the same temperature and interaction budget. This protocol aligns source information and interaction budgets while retaining differences in the methods' native interfaces; it does not equate token usage or monetary cost.

5.2 Main Results

Table 1 reports the two primary metrics. CRAFT leads on both, reaching 59.68 pair accuracy and 15.5 E2E accuracy on *Instruction-Only*, and 66.52 and 43.31 on *Multi-File*. Relative to VLM + HTML + Image, the strongest Instruction-Only baseline, the gains are 8.51 and 2.00 percentage points. On Multi-File, the best baseline differs by metric: OpenAI Agents SDK (OpenAI, 2026c) reaches 43.14 pair accuracy, while CrewAI reaches 18.90 E2E accuracy. The corresponding gains are 23.38 and 24.41 points.

Table 2: Ablation results (%). SALA: Structure-Aware Local Adjudication, PSR: Progressive Slot Re-grounding, Re.: Whole Reflection module, Pl.: planner.

	Instruction-Only		Multi-File	
Method	**Pair Acc.↑**	**E2E Acc.↑**	**Pair Acc.↑**	**E2E Acc.↑**
CRAFT	**59.68**	**15.5**	**66.52**	**43.31**
CRAFT w/o SALA	52.97	10.0	25.48	10.24
CRAFT w/o PSR	50.45	12.5	24.86	3.15
CRAFT w/o Re.	53.71	10.5	24.96	7.87
CRAFT w/o Re. & Pl.	51.63	12.5	24.17	6.30

Table 3: Results with GPT-5.4 mini (%).

	Instruction-Only		Multi-File	
Method	**Pair Acc.↑**	**E2E Acc.↑**	**Pair Acc.↑**	**E2E Acc.↑**
CRAFT	**41.57**	**9.5**	**60.66**	**9.45**
VLM + HTML + Image	34.06	8.5	25.86	5.95
OpenAI Agents SDK	29.24	2.0	49.91	3.54
SheetCopilot	12.75	0.5	0.66	0.0

The baseline rankings are consistent with different task bottlenecks. In the *Instruction-Only Track*, strong VLMs can recover many local label–value correspondences when the form layout is clearly exposed, but remain vulnerable to structural execution errors such as writing to nearby cells, misaligning values with slots, or overwriting template content. In the *Multi-File Track*, the challenge shifts upstream to evidence aggregation across heterogeneous sources, label alignment, and consistent multi-step execution, making general agent frameworks relatively more competitive. Existing spreadsheet agents still underperform both strong VLM baselines and general agents, likely because they are optimized for spreadsheet manipulation rather than fine-grained grounding and constraint-sensitive form filling. The tracks contain different form sets, so their absolute scores are not a controlled comparison of task difficulty.

5.3 Ablation and Backbone Transfer

Table 2 compares four component-removal variants. Removing SALA reduces pair accuracy by 6.71 points on *Instruction-Only* and 41.04 points on *Multi-File*. Removing Progressive Slot Re-grounding (PSR) reduces it by 9.23 and 41.66 points, respectively. The latter variant also lowers *Multi-File* E2E accuracy from 43.31 to 3.15. These results support structural adjudication and slot re-grounding as complementary parts of the evaluated pipeline.

Removing the entire reflection stage yields 53.71 and 24.96 pair accuracy across the two tracks. On Instruction-Only, its 3.26-point advantage over the variant without PSR (50.45) is consistent with repair becoming counterproductive without re-grounding. The corresponding Multi-File scores, 24.96 and 24.86, are nearly tied. Removing both reflection and planner-controlled execution yields 51.63 and 24.17. The non-monotonic E2E scores on *Instruction-Only* further indicate that the effects of individual components depend on the metric and setting; the full system provides the strongest overall results.

With GPT-5.4 mini (OpenAI, 2026b) as the backbone, CRAFT maintains the lead among the evaluated methods (Table 3). It reaches 41.57 pair accuracy on *Instruction-Only* and 60.66 on *Multi-File*, exceeding the strongest corresponding baselines by 7.51 and 10.75 points. This supports the relative benefit of the framework with a second backbone. Absolute performance remains model-dependent: Multi-File E2E accuracy is 9.45 with GPT-5.4 mini, compared with 43.31 with the default backbone.

5.4 Case Study

We further examine representative cases from both the ***Instruction-Only Track*** and the ***Multi-File Track***. Three recurring patterns emerge. *First*, vanilla VLM baselines remain competitive on visu-

ally regular layouts. For example, on the Left-Right Label-Slot case (Case 38), multiple rendering-based baselines achieve full success. However, their performance degrades on more irregular structures, such as the Mixed-Slot-Layout case (Case 29), where vanilla VLM baselines achieve only partial success. *Second*, spreadsheet-oriented agents show limited robustness across structurally diverse forms. Although SHEETCOPILOT (Li et al., 2023) succeeds on the Mixed-Section case (Case 88) and partially succeeds on the Left-Right Label-Slot case (Case 38), spreadsheet agents fail on most other cases, including the Repeated-Section case (Case 4), the Mixed-Slot-Layout case (Case 29), and the Bordered-Block case (Case 84). *Third*, the *Multi-File Track* remains challenging for nearly all baselines. On the Bordered-Block (Case 226), Mixed-Slot-Layout (Case 238), and Repeated-Section (Case 284) cases, most competing methods achieve only partial success or failure, indicating difficulties in consolidating evidence across files and grounding retrieved values back to the target form. Across all selected cases, `CRAFT` consistently achieves successful completion. Detailed case-level results and representative examples are provided in Appendix E.3.

6 RELATED WORK

Form and Document Understanding. Recent work studies form interaction and template-aware document processing through multimodal perception, including visual form agents (Toles et al., 2026; Li et al., 2025a), structured information extraction (Majumder et al., 2020; Cao & Luo, 2021), and template-based reconstruction or querying (Lin et al., 2025b;a; 2024; 2025c). FormGym's FieldFinder (Toles et al., 2026) and CommonForms (Barrow, 2026) make field localization a relevant point of comparison. `CRAFT` couples slot localization to spreadsheet repair: it restores overwritten template content, maps candidates to executable cell ranges, and protects reliable cells during refill.

Spreadsheet Agents. Spreadsheet agents such as SheetCopilot (Li et al., 2023), SheetAgent (Chen et al., 2025), and SheetBrain (Wang et al., 2026) support spreadsheet manipulation, planning, and validation. Reflection and validation are already part of this literature: SheetAgent uses iterative reflection, and SheetBrain validates execution results. Our contribution is the template-specific connection between diagnosis and repair: suspicious ranges carry restoration decisions and protection constraints into slot re-grounding and local refill.

General-Purpose LLM Agents. General frameworks such as LangGraph (LangChain, Inc., 2026), OpenAI Agents SDK (OpenAI, 2026c), CrewAI (CrewAI, 2026), and ReAct (Yao et al., 2022) enable multi-step reasoning and tool use, while Reflexion (Shinn et al., 2023), Self-Refine (Madaan et al., 2023), and CRITIC (Gou et al., 2024) study iterative refinement. `CRAFT` instantiates refinement with spreadsheet-specific spatial targets, template restoration, and edit-protection constraints.

7 CONCLUSION

Automatic spreadsheet form filling is essential in real-world enterprise settings. In this paper, we introduced an agent-based spreadsheet form filling system `CRAFT`. We developed *Form Filling Preprocessing* that transforms diverse multi-modal auxiliary sources into structured evidence via a dual-view spreadsheet representation. We proposed *Reflective Region Adjudication* that automatically detects and localizes structural violations and content inconsistencies to identify suspicious edits under template constraints. We proposed *Progressive Slot Re-grounding and Repair* that enhances filling accuracy by constraining agents to plausible writable slots while protecting reliable cells. Experimental results on `FormFillBench` demonstrated the effectiveness of our approach over baseline methods.

`CRAFT` is limited by the available source parsers and may require human clarification for ambiguous labels or multiple specifications within a cell. The benchmark is curated, and the reported point estimates cover two backbones and an LLM-based semantic judge. They establish performance under this evaluation protocol, not reliability across arbitrary enterprise workflows. Strict form-level completion remains challenging, especially on Instruction-Only forms and with the second backbone. Broadening evaluation coverage and quantifying run-to-run variability are directions for future work.

AI USE STATEMENT

Generative AI tools were used to assist with benchmark construction, including synthetic template generation and natural-language instruction generation, as described in Section 4. They were also used to support code development and improve the clarity and organization of the manuscript. LLMs and VLMs additionally serve as components of the evaluated systems and as the semantic judge, as described in Section 5.1. All AI-generated content and code were reviewed and validated by the authors, who take responsibility for the final benchmark, experiments, analyses, and claims.

ETHICS STATEMENT

This work does not involve human subjects or the collection of personal or sensitive individual information. The benchmark is constructed from synthetic or appropriately sourced documents, with source use and redistribution checked during curation. We acknowledge that automatic form filling may introduce risks related to privacy, erroneous automated entries, and downstream use in sensitive workflows. We therefore emphasize human verification of generated outputs and document the data construction and evaluation procedures to support responsible use.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide an anonymized code repository at `https://anonymous.4open.science/r/CRAFT-5715`, containing the implementation of `CRAFT`, the complete `FormFillBench` used in our experiments, prompts, and evaluation code.

REFERENCES

Anthropic. Introducing Claude Sonnet 4.6. `https://www.anthropic.com/news/claude-sonnet-4-6`, February 2026. Accessed: 2026-04-01.

Titus Barik, Kevin Lubick, Justin Smith, John Slankas, and Emerson Murphy-Hill. Fuse: a reproducible, extendable, internet-scale corpus of spreadsheets. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 486–489. IEEE, 2015.

Joe Barrow. Commonforms: A large, diverse dataset for form field detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1170–1179, 2026.

Rongyu Cao and Ping Luo. Extracting zero-shot structured information from form-like documents: Pretraining with keys and triggers. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 12612–12620, 2021.

Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, Jianye Hao, Hangyu Mao, and Fuzheng Zhang. Sheetagent: towards a generalist agent for spreadsheet reasoning and manipulation via large language models. In *Proceedings of the ACM on Web Conference 2025*, pp. 158–177, 2025.

Clark County, Washington. It vendor cybersecurity questionnaire. `https://clark.wa.gov/sites/default/files/2025-06/rfp-927-exhibit-b_it_vendor_cybersecurity_questionnaire.xlsx`, 2025. Official spreadsheet compliance questionnaire, accessed 2026-04-10.

CrewAI. Crewai documentation. `https://docs.crewai.com/`, 2026. Accessed: 2026-03-31.

Wensheng Dou, Liang Xu, Shing-Chi Cheung, Chushu Gao, Jun Wei, and Tao Huang. Venron: a versioned spreadsheet corpus and related evolution analysis. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 162–171, 2016.

Google. Gemini 3.1 flash-lite-preview. `https://ai.google.dev/gemini-api/docs/models/gemini-3.1-flash-lite-preview`, 2026. Google AI for Developers documentation. Accessed: 2026-04-01.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, volume 2024, pp. 57734–57811, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pp. 32808–32824, 2024.

LangChain, Inc. Langgraph documentation. https://docs.langchain.com/oss/python/langgraph/overview, 2026. Accessed: 2026-03-31.

Bobo Li, Yuheng Wang, Hao Fei, Juncheng Li, Wei Ji, Mong-Li Lee, and Wynne Hsu. Formfactory: An interactive benchmarking suite for multimodal form-filling agents. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 13273–13280, 2025a.

Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhao-Xiang Zhang. Sheetcopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems*, 36:4952–4984, 2023.

Jia-Nan Li, Jian Guan, Wei Wu, Zhengtao Yu, and Rui Yan. 2d-tpe: Two-dimensional positional encoding enhances table understanding for large language models. In Guodong Long, Michale Blumestein, Yi Chang, Liane Lewin-Eytan, Zi Helen Huang, and Elad Yom-Tov (eds.), *Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025*, pp. 2450–2463. ACM, 2025b. doi: 10.1145/3696410.3714920. URL https://doi.org/10.1145/3696410.3714920.

Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, and Aditya G. Parameswaran. Towards accurate and efficient document analytics with large language models. *arXiv preprint arXiv:2405.04674*, 2024. To appear in SIGMOD 2025.

Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G Parameswaran. Twix: Automatically reconstructing structured data from templatized documents. *arXiv preprint arXiv:2501.06659*, 2025a.

Yiming Lin, Mawil Hasan, Rohan Kosalge, Alvin Cheung, and Aditya G Parameswaran. Visual template inference for data extraction from documents. *Proceedings of the ACM on Management of Data*, 3(6):1–27, 2025b.

Yiming Lin, Madelon Hulsebos, Ruiying Ma, Shreya Shankar, Sepanta Zeighami, Aditya G Parameswaran, and Eugene Wu. Querying templatized document collections with large language models. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pp. 2422–2435. IEEE, 2025c.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Bodhisattwa Prasad Majumder, Navneet Potti, Sandeep Tata, James Bradley Wendt, Qi Zhao, and Marc Najork. Representation learning for information extraction from form-like documents. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6495–6504, misc, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.580. URL https://aclanthology.org/2020.acl-main.580/.

OpenAI. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/, March 2026a. Accessed: 2026-09-26.

OpenAI. Introducing GPT-5.4 mini and nano. https://openai.com/index/introducing-gpt-5-4-mini-and-nano/, March 2026b. OpenAI Index page, accessed 2026-04-18.

OpenAI. Openai agents SDK documentation. https://developers.openai.com/api/docs/guides/agents-sdk/, 2026c. Accessed: 2026-03-31.

Sajjadur Rahman, Kelly Mack, Mangesh Bendre, Ruilin Zhang, Karrie Karahalios, and Aditya Parameswaran. Benchmarking spreadsheet systems. In *Proceedings of the 2020 acm sigmod international conference on management of data*, pp. 1589–1599, 2020.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *International conference on learning representations*, volume 2025, pp. 98190–98243, 2025.

Matthew Toles, Isaac Song, Rattandeep Singh, and Zhou Yu. Formgym: Doing paperwork with agents. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3771–3785, 2026.

University of California, Berkeley. Domestic travel expense reimbursement form. `https://www.erso.berkeley.edu/web/sites/default/documents/Continental%20Reimbursement%20Form.xlsx`, 2026. Official spreadsheet reimbursement form, accessed 2026-04-10.

Ziwei Wang, Jiayuan Su, Mengyu Zhou, Huaxing Zeng, Mengni Jia, Xiao Lv, Haoyu Dong, Xiaojun Ma, Shi Han, and Dongmei Zhang. Sheetbrain: A neuro-symbolic agent for accurate reasoning over complex and large spreadsheets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33800–33808, 2026.

Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in neural information processing systems*, 34:12077–12090, 2021.

Liang Xu, Wensheng Dou, Chushu Gao, Jie Wang, Jun Wei, Hua Zhong, and Tao Huang. Spreadcluster: recovering versioned spreadsheets through similarity-based clustering. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 158–169. IEEE, 2017.

Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1192–1200, 2020.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Wenwen Yu, Ning Lu, Xianbiao Qi, Ping Gong, and Rong Xiao. Pick: processing key information extraction from documents using improved graph learning-convolutional networks. In *2020 25th International conference on pattern recognition (ICPR)*, pp. 4363–4370. IEEE, 2021.

Ruiyan Zhu, Xi Cheng, Ke Liu, Brian Zhu, Daniel Jin, Neeraj Parihar, Zhoutian Xu, and Oliver Gao. Sheetmind: An end-to-end llm-powered multi-agent framework for spreadsheet automation. *arXiv preprint arXiv:2506.12339*, 2025.

A TASK FORMALISM

In this section, we first introduce the elements of spreadsheet forms. Then we give the definition of spreadsheet form filling.

A.1 SPREADSHEET FORMS

Form Elements. We first define the core elements that determine how information is organized in a spreadsheet form and where values should be written.

• ***Form Range.*** A spreadsheet form is inherently structurally grounded. A *range* R is an explicit extent in a spreadsheet form, formally represented as a set of contiguous units $R = \{u_1, u_2, \ldots, u_m\}$. In our formulation, each unit u_j is a *cell* $c = (pos, val, fmt)$ consisting of a position pos (e.g., A1), a value val (e.g., text, number), and formatting attributes fmt (e.g., color, font, border style).

• ***Form Label.*** Following the common paired formulation used in document understanding (Xu et al., 2020) and key information extraction (Yu et al., 2021), a *label* represents the semantic descriptor of a fillable target in a spreadsheet form, such as "Name" or "Department". To distinguish repeated labels (e.g., multiple "Name" entries in duplicated sections), we define a composite *label identifier* $\ell = (\text{label}, \text{label_num})$, where label is the label text and label_num indexes repeated occurrences of the same label. If a label appears only once, label_num $= 0$.

• ***Label–Value Pair.*** A *label–value pair* is the fundamental semantic–structural unit in a spreadsheet form. Unlike JSON or relational tables, the correspondence between a label and its writable range in a spreadsheet form is often implied through layout cues such as alignment and proximity. Specifically, a label–value pair is defined as $p = (\ell, R, \tau)$, where ℓ is the label identifier, R denotes the associated writable range for filling, and $\tau \in \mathcal{T}$ represents the local structural relation between the label and its target range (e.g., left–right alignment or top–down alignment).

Definition A.1 (Spreadsheet Form)**.** A spreadsheet form $\mathcal{F}$ is a layout-rich spreadsheet object designed to organize and collect structured information. It is formally defined as a tuple $\mathcal{F} = (\mathcal{P}, \mathcal{C}, \mathcal{T})$, where:

- $\mathcal{P}$ is the set of *label–value pairs* that define the core fillable targets.
- $\mathcal{C}$ is the set of *auxiliary semantic contents* (e.g., section titles, instruction texts, and notes). These elements do not correspond to fillable targets, but provide important semantic and structural cues for interpreting the spreadsheet form.
- $\mathcal{T}$ captures the positional and structural relations among elements, including relative positions, grouping relations, hierarchical structures, and form-level layout organizations across spreadsheet regions.

A.2 FORM FILLING TASK

Given a template spreadsheet form $\mathcal{F}$, an input text $\mathcal{X}$ (typically a natural language description), and a set of auxiliary source files $\mathcal{S}$, the spreadsheet form filling task aims to accurately extract and consolidate relevant information from $\mathcal{X}$ and $\mathcal{S}$, then write it into the corresponding spatial regions in $\mathcal{F}$.

Definition A.2 (Spreadsheet Form Filling)**.** The spreadsheet form filling task aims to produce a set of filled entries $\mathcal{O}$ that specifies what value should be written into which range of the spreadsheet form, based on the input text $\mathcal{X}$ and auxiliary source files $\mathcal{S}$. Formally:

$$\mathcal{O} = \{e_1, e_2, \ldots, e_N\} \tag{2}$$

where each *filled entry* $e_i = (\ell_i, v_i, R_i)$ specifies the value v_i to be written into range R_i for label identifier ℓ_i. Labels not supported by $\mathcal{X}$ or $\mathcal{S}$ are assigned $v_i = \text{null}$. Applying $\mathcal{O}$ to the spreadsheet form $\mathcal{F}$ yields the final filled spreadsheet.

Example 4 (Duplicated Labels)**.** As illustrated in Figure 1, the business card order spreadsheet form contains two repeated labels "Name". They are identified as $\ell_1 = (\text{Name}, 1)$ and $\ell_2 = (\text{Name}, 2)$.

According to the provided source files, the resulting filled entries would be

$$e_1 = (\ell_1, \text{“Robert J. Miller”}, \{\text{F4}\}),$$
$$e_2 = (\ell_2, \text{“Robert J. Miller, Ph.D.”}, \{\text{B20}\}).$$

As shown in the examples, the spreadsheet form filling task is inherently complex and distinguished from standard information extraction. First, spreadsheet forms often do not expose explicit schema-level label–value mappings, requiring models to infer implicit structure from layout signals (e.g., spatial alignment and layout grouping) to correctly associate labels with their corresponding writable regions. Second, correct completion is often context-dependent: auxiliary contents $\mathcal{C}$, such as instructions, section headers, or formatting constraints, may affect both what should be filled and how it should be written. Third, spreadsheet form filling requires exact range grounding, since predicted values must be written into the correct cells or ranges; even minor misplacement may overwrite template elements (e.g., labels or auxiliary contents) or the structural organization of the spreadsheet form.

A.3 Evaluation Criteria

Let $e_i = (\ell_i, v_i, R_i)$ be a predicted entry and $e_i^* = (\ell_i^*, v_i^*, R_i^*)$ be the corresponding ground-truth entry. We evaluate correctness at the entry level. A predicted entry is considered correct if both the semantic value and the target range match the ground truth:

$$\text{Correct}(e_i, e_i^*) = \mathbb{I}[\text{MatchValue}(v_i, v_i^*) \wedge (R_i = R_i^*)]\,. \tag{3}$$

Here, MatchValue allows fuzzy matching between semantically equivalent textual values. This is important, because in spreadsheet form filling, semantic correctness is often more critical than exact string equality. For example, the same value may appear in different surface forms, such as alternative date formats (e.g., “2024-03-15” vs. “March 15, 2024”). In addition, if a value spans multiple cells or lines, the contents of the corresponding range are concatenated before comparison.

B Pilot Study Details

Before presenting our method, we conduct a pilot study to isolate the dominant failure patterns of general-purpose LLMs in spreadsheet form filling. To evaluate how representative systems handle structural grounding without conflating it with broader task complexity, we build a controlled pilot set. We randomly select 15 forms from two common structural categories (*Mixed-Slot-Layout* and *Repeated-Section Forms*) and manually expand them into 57 instances by keeping the filling semantics fixed while varying the relative spatial position between each label and its target slot.

• ***Sensitivity to Relative Label–Slot Layouts.*** Figure 2 shows that label–slot direction affects performance, with different sensitivity across systems. The left-label/right-slot arrangement is generally favorable, but OpenAI Agents SDK also performs strongly when the slot is below the label (0.842, compared with 0.858 on right-slot layouts). The pilot motivates attention to local correspondence rather than a claim that every noncanonical layout is difficult.

• ***Unstable but Structured Error Patterns.*** Beyond aggregate accuracy drops, we observe that failed outputs exhibit consistent, structured qualitative patterns. A single incorrectly filled form often contains multiple local errors concentrated in nearby but structurally incorrect regions. Common failure modes include writing directly into the label cell itself, filling an adjacent cell on the wrong side of the label, or generating locally plausible but role-inconsistent entries. Furthermore, repeated executions on the same form yield unstable outputs, shifting the error locations across runs.

In summary, reliable spreadsheet form filling cannot be treated as unconstrained one-shot generation. The observed layout sensitivity and structured error patterns motivate a localized repair process that explicitly validates edits against template structure and re-grounds content using local geometric constraints.

C Implementation Details and Worked Examples

C.1 Form Filling Preprocessing

Existing systems often flatten structured inputs into one-dimensional sequences, which can obscure spatial relationships and weaken downstream reasoning over tables and form-like artifacts (Xu et al., 2020; Li et al., 2025b). Moreover, the evidence required for form filling is often scattered across heterogeneous auxiliary sources, such as emails, PDFs, spreadsheets, and other semi-structured records. Existing general-purpose agent frameworks (LangChain, Inc., 2026) and spreadsheet agents (Li et al., 2023) do not explicitly provide a unified representation that preserves the source-specific structural and spatial cues required in our form-filling setting. The core challenge is therefore to convert heterogeneous auxiliary sources into a unified yet structure-aware evidence representation that supports downstream form filling. To address this challenge, we design a preprocessing procedure that (1) represents the template spreadsheet form to provide an executable space and a layout-aligned visual modality enhancement, (2) transforms auxiliary sources into structured evidence representations and (3) performs an initial form pre-filling, thereby laying the foundation for subsequent refinement and precise re-filling.

C.1.1 Dual-View Spreadsheet Representation

CRAFT first parses the spreadsheet template to recover cell coordinates, and determines the exact form range from the maximal populated row–column extent. Then, CRAFT derives two complementary views over this range:

• **(1) Executable Spreadsheet Representation.** Over the identified form range, CRAFT extracts both semantic and formatting attributes (e.g., cell content, fill color, and font properties), and constructs an executable spreadsheet representation from the raw spreadsheet data. In addition, CRAFT derives a structural summary that captures merged-cell regions, writable and non-writable areas, and form-specific constraints such as dropdown validations together with their admissible candidate values. *This representation enables the system to operate directly on constrained spreadsheet regions instead of an ambiguous flattened text view.*

• **(2) Layout Information Enhancement.** Because the executable spreadsheet representation mainly captures the form as a one-dimensional programmatic structure, it is insufficient for later reflection and validation stages that require explicit layout understanding, such as layout consistency checking and suspicious-range localization. To support these later stages, CRAFT further augments the spreadsheet form with a layout view. Specifically, it first parses the spreadsheet into layout-aligned structured markup, then renders this markup into a sheet image, and records the alignment between spreadsheet coordinates, markup regions, and rendered regions. *This layout-aligned representation preserves the layout organization of the form and links each cell region to its corresponding layout area.*

C.1.2 Source File Preprocessing

To better preserve the semantic and structural information in source files for downstream form filling, CRAFT preprocesses these source files in three steps:

• **(1) Type-Based Content Extraction.** CRAFT first processes each file independently according to its type. (i) For *text and markup* documents, the content is organized into chapters, with evidence spans preserved as textual grounding units. (ii) *Structured files* (e.g., JSON, XML) are normalized into label–value records , preserving the original label hierarchy whenever available. (iii) *Spreadsheets and tables* are represented by their column headers together with row-level records extracted from the table body. (iv) *Emails and HTML pages* are decomposed into header slots (sender, subject, date), body chapters, and embedded tables when present. (v) *PDFs and scanned files* are normalized into layout-organized text blocks, detected tables, and layout anchors used for later grounding.

• **(2) Structure-Based Evidence Formatting.** With the extracted file contents, CRAFT further calculates a set of lightweight structural signals to characterize how the extracted content is organized, including table density, label–value regularity, chapter patterns, and repetition strength. Concretely, table density measures the fraction of extracted content tokens that belong to detected row–column regions among all extracted tokens in the file; label–value regularity measures how consistently text

blocks can be paired into aligned label–value patterns with repeated relative offsets; section-pattern strength captures the frequency of title-like lines followed by grouped descendant content; and repetition strength captures the recurrence of records or rows sharing the same local schema. CRAFT then uses these signals to determine which structural pattern is most salient in the file and standardizes the extracted content accordingly. For example, (i) high label–value regularity yields a flat label–value view; (ii) prominent chapter patterns induce sectioned blocks; (iii) strong row–column regularity or repeated-item structure leads to record- or table-oriented organization; (iv) deeply nested relations are preserved as hierarchical JSON-like structures; and (v) files exhibiting both discourse and tabular characteristics are represented in a hybrid chapter-and-table form. For weakly structured sources, CRAFT retains evidence-grounded text spans as the primary representation.

• **(3) Source Bundle Assembling.** The processed results are then assembled into a unified source bundle that contains both normalized content and compact structural summaries, which is provided as a part of the prompt to the planner in the subsequent prefilling stage introduced in Section C.1.3.

C.1.3 Planner-Guided Prefilling

With the carefully processed form template and source files, CRAFT next utilizes a planner-guided prefill agent to conduct the initial form prefilling, whose output is further refined in the following stages (see Section C.2).

• **(1) Structure-Based Filling Plan Generation.** Given the structure-centered representation of the spreadsheet form extracted in Section C.1.1, CRAFT first generates a structural summary by extracting key structural elements such as merged cells and label cells. Then we prompt the planning agent (by concatenating the summary, user instruction, source bundles) to generate a detailed prefilling plan: (1) It may specify planner-selected background reads that associate a target form region with the most relevant source bundles, or provide short operational notes to indicate how the region should be filled as shown in Figure 3; (2) It also specifies global filling policies, such as whether unsupported or unspecified slots should be left blank according to the user's instruction.

• **(2) Plan-Guided Prefill Execution.** The prefill agent receives the user instruction, the structured source bundle, the planner-derived policy, and the canonical form workspace as input. Under the generated prefilling plan, it fills the template spreadsheet form programmatically using the provided Excel skills.

To keep our prefill process scalable to relatively large spreadsheet forms, we adopt a fallback mechanism. When the canonical workbook is too large to be safely inlined into the model context, CRAFT switches to a compact tool-first execution mode instead of forcing a full-context prompt. In this fallback mode, the agent is instructed to first inspect the workbook structure itself, and then read only small local ranges that are directly relevant to the current filling decision.

C.2 Reflective Region Adjudication

The prefilled form often contains many errors, such as overwriting labels instead of their corresponding slots, placing values into adjacent but incorrect cells, or violating form-specific constraints (e.g., dropdown validation rules). These errors are often difficult to detect because many incorrect filling actions appear plausible in isolation. For instance, a value written into a neighboring cell may seem reasonable, while still breaking the intended local label–value alignment. Such errors can substantially degrade downstream form-filling accuracy (Stechly et al., 2025; Huang et al., 2024). To address this issue, in this section we introduce (1) a local reflective, evidence-grounded validation mechanism that identifies reliable and suspicious cells, along with (2) a range construction procedure that further consolidates them into compact suspicious regions for downstream repair.

C.2.1 Evidence-Grounded Filling Validation

To support reliable correction, we first perform an evidence-grounded validation process to identify suspicious cells that likely require repair and reliable cells that need to be protected. Rather than relying on unconstrained LLM self-reflection (Stechly et al., 2025; Huang et al., 2024), this process is explicitly grounded in the dual-view spreadsheet workspace constructed during preprocessing. The *layout-aligned view* supports layout consistency checking against the original template, while the *executable spreadsheet representation* exposes structural cues such as writable regions, static

Table 4: Representative risk signals used in structure-aware local adjudication. These examples illustrate template-integrity and layout-grounding risks.

Risk Signal	Illustrative Example
Label-cell overwrite	The value "11-2045-5-601002-4310" is written into merged range `K19:T19`, which originally contains the static template label "Account Number", thereby overwriting template-side content.
Semantic ambiguity	The value "Robert J. Miller" is written into cell `B20`, even though this slot should be associated with the first occurrence of `Name` rather than the second one, creating ambiguity between duplicated label groups.
Template inconsistency	Cell `K20` is left-aligned, while the neighboring filled cells in the same local group are all center-aligned, indicating a local formatting inconsistency.
Label–value layout abnormality	The slot for `Signature` is located above its label, but the value is written into a nearby cell below or beside the label, violating local top-slot / bottom-label layout.

template cells, and slot-level constraints. The goal of this stage is therefore to determine which edits are potentially problematic and to attach sufficient diagnostic evidence for downstream correction.

The design of this validation module is guided by two principles. *(1) Dual-view-grounded validation.* Suspicious edits are assessed jointly from layout evidence and structural execution semantics, rather than from free-form model introspection alone. *(2) Decoupled discovery and adjudication.* Reflective Region Adjudication adopts a two-stage pipeline: the first stage performs broad screening to identify potentially problematic regions with high recall, and the second stage conducts localized diagnosis to determine the exact error type and validation state of each flagged location. This staged design allows downstream refill to operate over explicit repair targets instead of blindly regenerating the workbook.

• **Stage 1: Layout-Grounded Global Screening.** The first stage performs a coarse scan over the entire prefilled workbook to identify *reliable* and *suspicious* cells. To achieve this, `CRAFT` uses the *layout-aligned view* (Section C.1.1) to compare the pristine template snapshot with the current prefilled workbook. By feeding these aligned layout views, together with the user instruction and edit history, into a VLM, layout-grounded global screening assesses whether the overall organization has been disrupted, for example due to misplaced or missing values, abnormal overwrites, or inconsistent local alignment.

• **Stage 2: Structure-Aware Local Adjudication.** The second stage complements the coarse candidates identified in Stage 1 by performing localized diagnosis over the *executable spreadsheet representation*. Its goal is to determine whether each non-empty cell pattern exhibits any typical error mode observed in our pilot study (Section 2).

Structure-Aware Local Adjudication evaluates suspicious locations using two categories of spreadsheet-specific risk signals. *(1) Template-integrity risks* are checked programmatically to detect whether a generated value has overwritten a static template cell, such as a label, title, or header. *(2) Layout-grounding risks* are assessed by the VLM to capture structurally implausible fills, such as inverted label–value ordering or abnormal separation between a label and its associated value region. Representative risk signals are summarized in Table 4. A cell is marked as *suspicious* only when it triggers at least two risk signals. This conservative threshold is intended to limit repairs triggered by a single noisy cue. As described in Section 3.2, global screening can also contribute a repair candidate.

C.2.2 Repair Range Construction

The identified *suspicious* cells are not passed downstream in isolation. Instead, `CRAFT` expands and consolidates them into compact *suspicious ranges*, so that subsequent correction operates over coherent local regions rather than individual cells. This design avoids two undesirable extremes: repairing single cells would discard the surrounding layout context needed for re-grounding and refill, while regenerating the entire form would unnecessarily rewrite reliable regions and amplify error propagation.

Concretely, after combining the cell-level judgments from layout-grounded global screening and structure-aware local adjudication, CRAFT expands suspicious cells in two stages to construct suspicious ranges. It first expands them in spreadsheet space, where merged cells are treated as atomic cells, so as to preserve sufficient local semantic and structural context. It then further expands the corresponding rendered regions in pixel space with a small margin, so that cell-border information and surrounding layout cues are also retained for downstream repair.

Each repair range further carries its adjudication evidence, including the detected error types and whether the range is associated with explicit template-integrity violations (e.g., `label-cell overwrite`), other structure-aware risk signals, or only a coarse warning from the global screening stage. This information directly determines the downstream repair strategy. For example, if Stage 2 reports a template-integrity risk, the refill module first triggers *template restoration* before attempting to write new values. In contrast, ranges associated only with layout-grounding risks or content anomalies may require only localized refilling.

Finally, the planner determines whether each constructed suspicious range should enter the repair stage. Together, these outputs form a structured repair specification consisting of localized repair ranges, their diagnostic provenance, restoration indicators when needed, reliable cells, and cell-level validation states, which serves as the interface to the downstream repair module.

C.3 Progressive Slot Re-grounding and Repair

Although Reflective Region Adjudication identifies suspicious ranges, reliable repair still requires a controlled local correction process rather than ordinary VLM-based refill within those ranges. The system must determine whether overwritten local template structure should first be restored, where revised values should be written, what nearby content must be preserved, and how local updates are applied without disturbing the surrounding structure. To address this, we introduce the *Guided Refill* module, which turns prior validation outputs into constrained local repair actions while preserving reliable context. As shown in Figure 3, Guided Refill proceeds in four stages.

• **Step 1: Template Restoration.** Before guided refill, some suspicious ranges may need *template restoration* if earlier execution has overwritten static template content. To detect this, CRAFT compares the current mutable workbook against the immutable template snapshot at the cell level, checking cell values, cell styles, and merged-range structure. When overwriting is detected, CRAFT restores the affected range to its original template state while preserving cells that have already been marked as reliable. This allows refill to proceed from a clean structural context and helps prevent earlier structural overwriting from propagating into later updates.

• **Step 2: Rectangle-Aware Slot Grounding.** After template restoration, local repair still requires accurate grounding of the writable slots within each suspicious range. This remains difficult for general-purpose vision-language agents, which have limited reliability in precise slot localization and require dedicated localization assistance (Toles et al., 2026). We therefore introduce a dedicated pretrained model, the *Rectangle-Aware Slot Grounder (RASG)*, to provide more stable candidate writable slots for subsequent local refill.

Model Design. RASG is designed to localize writable cells by exploiting the rectangular geometry commonly exhibited by form slots. As illustrated in Figure 4, it adopts a lightweight SegFormer-B1 (Xie et al., 2021) backbone with three task-specific prediction heads. Specifically, given the layout view of a spreadsheet form, RASG first encodes the image through four hierarchical stages. The resulting multi-scale features are then linearly projected, upsampled, and concatenated into a shared decoder representation for all three heads. The main head predicts the final rectangle logits, while the two auxiliary heads predict horizontal and vertical boundary cues, respectively. The rectangle logits are then rectified and reconstructed using the boundary predictions with a set of empirically chosen parameters. Finally, the reconstructed rectangle masks are aligned with candidate writable cells using area-based thresholds. By explicitly modeling cell-like rectangular geometry, RASG narrows the repair space to plausible writable slots and provides more task-aligned grounding for subsequent local refill.

Training Strategy. RASG is trained with a three-stage strategy, which progressively increases data realism while preserving high slot recall. First, we train on synthetic spreadsheet forms with both bounding-box- and cell-level supervision, which allows RASG to learn basic label–slot layout

patterns. In the second stage, we continue training on CommonForms (Barrow, 2026), a PDF-based dataset that provides only bounding-box supervision. This stage exposes RASG to more realistic rendering noise and layout variation, improving its robustness under weaker supervision. In the third stage, we further adapt the model on a manually filtered and annotated subset of FUSE (Barik et al., 2015), which provides both bounding-box- and cell-level supervision for real-world spreadsheet forms.

When no reliable slot is available, the controller falls back to guarded local repair using only the suspicious range, the template summary, and the protection policy.

• **Step 3: Local Label–Value Matching.** After RASG proposes a set of candidate writable cells, this stage converts these candidates into structured refill hints for subsequent local repair. In particular, we use a VLM to determine whether each candidate is a valid value cell and, if so, which nearby label it should be associated with.

Candidates judged valid are converted into positive hints that identify plausible value cells for a label, while invalid candidates are converted into negative hints that explicitly rule out non-target locations. In addition, `CRAFT` may retain weaker context-only hints that preserve nearby label context when a specific target cell cannot yet be selected. Together, these hints turn local refill from unconstrained generation into a more guided local repair process, narrowing the feasible write space while preserving useful local context under ambiguity.

• **Step 4: Guided Refill Execution.** Using the original prompt, the range-level refill hints, the restored local spreadsheet state, and feedback from previous validation and repair steps, `CRAFT` first constructs an explicit local refill plan that specifies where repair is allowed, which locations must be reconsidered, and which cells must remain protected. Under this plan and guided by the parsed hints, the refill agent performs local refill by generating and applying spreadsheet-editing actions to suspicious cells while leaving protected cells unchanged.

C.4 Qualitative Visualization of RASG

Figure 5 visualizes the intermediate directional cues and final slot predictions produced by RASG on representative form layouts. The horizontal and vertical cues capture complementary slot boundaries, which are subsequently combined with the segmentation output to reconstruct rectangular writable regions. The examples illustrate how RASG represents candidate slots under diverse layouts, including adjacent fields, elongated regions, and irregularly arranged form sections.

D Benchmark Construction and Supplementary Metrics

Despite rapid progress in LLM-based agents for document interaction and spreadsheet automation (Lin et al., 2025c; Chen et al., 2025), existing form filling benchmarks mainly target web-based or document-centered settings (Li et al., 2025a; Toles et al., 2026), while existing spreadsheet evaluation efforts focus on spreadsheet understanding and manipulation rather than spreadsheet form filling (Rahman et al., 2020). To our knowledge, spreadsheet form filling has not been explicitly studied as a dedicated benchmark task. To address this gap, we introduce `FormFillBench`.

Benchmark Overview. `FormFillBench` is a benchmark for spreadsheet form filling, organized into two tracks to balance comparability and realism. ***(1) Instruction-Only Track.*** In this controlled setting, an agent fills a spreadsheet form from a natural-language instruction alone. ***(2) Multi-File Track.*** In this more realistic setting, an agent must consolidate evidence from one or more supporting files under a standardized prompt and complete the target form correctly.

Benchmark Construction. The spreadsheet form templates are collected from FUSE (Barik et al., 2015), VEUSES (Xu et al., 2017), and Venron2 (Xu et al., 2017; Dou et al., 2016), together with publicly collected online templates and additional synthetic templates generated with Claude Sonnet 4.6 (Anthropic, 2026).

For the ***Instruction-Only Track***, each spreadsheet form is paired with a synthesized natural-language instruction that specifies only the values to be filled, without revealing their target cell coordinates or explicit spatial relations in the form. Using the rendered form image and key annotations, we employ Gemini 3.1 Flash-Lite-Preview (Google, 2026) to generate these value-only instructions. All

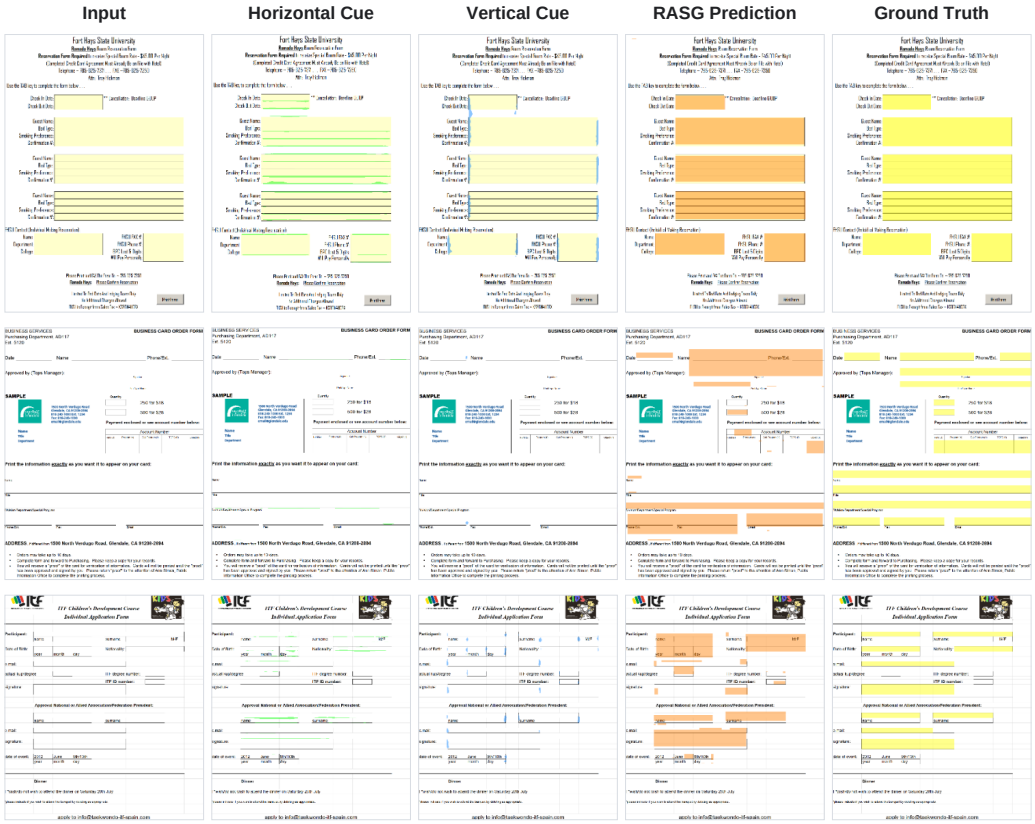

Figure 5: Qualitative visualization of RASG on representative spreadsheet forms. From left to right, we show the rendered input, horizontal boundary cues, vertical boundary cues, reconstructed RASG predictions, and ground-truth writable regions.

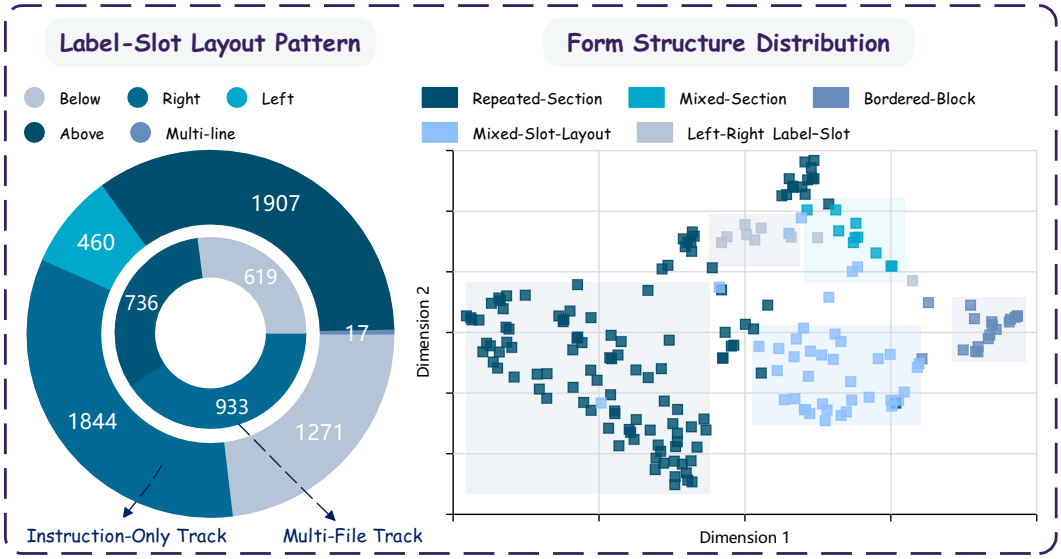

Figure 6: Statistics and Structural Diversity of `FormFillBench`.

templates are normalized into a unified spreadsheet representation. This track primarily emphasizes the spreadsheet-side form filling difficulty: even when the required information is already given in

the instruction, agents may still fail to place values into the correct fields under complex and diverse spreadsheet layouts.

For the ***Multi-File Track***, each form is paired with several supporting files and a standardized prompt (e.g., "Fill the form based on the provided source files."), which provides no field values or spatial hints, requiring the agent to use only explicitly supported information. The supporting files are drawn from open-source resources and synthetic enterprise document collections, spanning diverse departments, business scenarios, and heterogeneous file types such as emails, tables, notes, and semi-structured records. Compared with the Instruction-Only Track, this track places more emphasis on finding, consolidating, and organizing evidence across heterogeneous files before grounding the retrieved values back into the target form.

Synthetic templates are used to broaden structural coverage rather than to replace real-source forms. We therefore use them as a complement to public and corpus-derived templates, and all candidate samples undergo automated consistency checks followed by manual verification before inclusion.

Benchmark Statistics. `FormFillBench` contains 327 form instances in total: Among them, 200 samples with 5,499 label–value pairs belong to the ***Instruction-Only Track***, while 127 samples with 2,288 pairs belong to the ***Multi-File Track***. The ***Multi-File Track*** spans 11 file types, dominated by TXT (177), CSV (154), Markdown (103), JSON (95), and EML (80), with a longer tail, including YAML, presentation files, spreadsheets, images, and PDF files. Each Multi-File case is typically accompanied by around five supporting files.

Beyond source diversity, `FormFillBench` also covers diverse local label–slot layouts, including *above*, *below*, *left*, *right*, and *multi-line* patterns. As shown in Figure 6, both tracks exhibit substantial layout diversity, although the ***Multi-File Track*** is more concentrated in *above*, *below*, and *left* patterns.

Template-Level Structural Diversity. To summarize template-level structural diversity in `FormFillBench`, we derive five coarse archetypes from feature-based clustering over layout characteristics. *Left-Right Label–Slot Forms* follow a canonical left-label/right-value arrangement for most fields. *Mixed-Slot-Layout Forms* remain organized around label–value fields, but the relative placement of labels and values varies across regions. *Repeated-Section Forms* consist of a vertical sequence of sections that repeat similar field-entry patterns. *Mixed-Section Forms* combine several distinct sections with different internal layouts across the page. *Bordered-Block Forms* are organized around prominent bordered or table-like blocks. These archetypes are intended as an interpretable summary of benchmark-level structural diversity rather than a strict taxonomy.

Evaluation Metrics. We evaluate form-filling performance using two common metrics across both tracks, together with track-specific supplementary metrics.

(1) Pair Accuracy. Pair Accuracy measures the proportion of target pairs for which both the filled value and the target cell or range are correct. It provides a fine-grained view of form-filling quality.

(2) End-to-End Accuracy. End-to-End Accuracy measures the fraction of forms for which every target pair is correct, including fields that should remain blank. It is therefore a strict form-level metric.

(3) Fill Precision, Recall, and F1 for the Instruction-Only Track. To distinguish underfilling from overfilling, we additionally evaluate the fill-versus-blank decision for each target field, independent of value correctness. We report Fill Precision, Recall, and F1 based on true positives (non-null fields that are filled), false negatives (non-null fields left blank), and false positives (null fields that are nevertheless filled).

(4) Fill Accuracy for the Multi-File Track. Since all target fields in the Multi-File Track are intended to be filled, we additionally report Fill Accuracy to measure the fraction of target fields that are filled by the system, regardless of value correctness.

Table 5: Complete benchmark results on the Instruction-Only and Multi-File tracks. All metrics are higher-is-better.

Method	Instruction-Only Track					Multi-File Track		
	Pair Acc. ↑	E2E Acc. ↑	Rec. ↑	Prec. ↑	F1 ↑	Pair Acc. ↑	E2E Acc. ↑	Fill Acc. ↑
Vanilla VLM baselines								
VLM+HTML+Image	51.17	13.5	59.97	95.94	73.80	32.60	12.60	39.25
VLM+HTML	49.26	11.0	57.37	95.96	71.80	26.31	7.87	32.34
VLM+Excel Skills+Coding	10.66	0.0	**80.91**	7.93	14.44	0.26	0.0	0.93
General agent frameworks								
LangGraph	44.28	10.0	56.19	95.54	70.76	33.65	12.60	40.82
OpenAI Agents SDK	45.64	9.5	57.65	95.94	72.02	43.14	16.54	50.79
CrewAI	41.77	9.5	51.96	95.73	67.36	40.25	18.90	43.44
ReAct	41.66	11.0	52.45	95.56	67.73	34.00	13.39	38.81
Spreadsheet agents								
SheetAgent	8.89	0.0	2.54	58.41	4.87	0.26	0.0	0.00
SheetBrain	10.98	0.0	3.58	65.92	6.78	0.26	0.0	0.00
SheetCopilot	33.02	7.5	37.94	93.82	54.03	0.26	0.0	1.09
SheetMind	10.69	0.0	3.13	63.37	5.96	0.26	0.0	0.00
Ours								
CRAFT	**59.68**	**15.5**	71.08	**96.52**	**81.87**	**66.52**	**43.31**	**90.87**

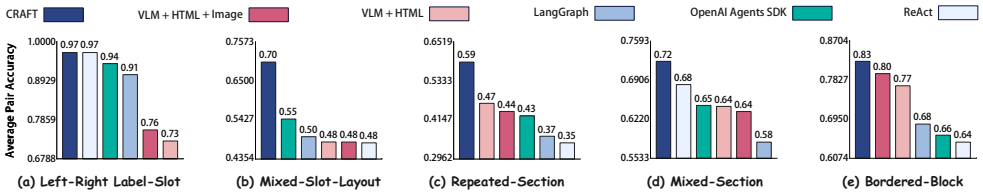

Figure 7: Comparison of average pair accuracy across five representative form types for CRAFT and the top-5 baselines. The reported scores are computed by first measuring pair accuracy on individual forms and then averaging within each form type.

E Supplementary Results and Analysis

E.1 Complete Benchmark Scores

Table 5 complements the primary metrics in Table 1 with fill precision, recall, F1, and Fill Accuracy, following the definitions in Appendix D.

E.2 Evaluation Across Spreadsheet Form Types

Figure 7 reports average pair accuracy across five representative form types. CRAFT is top-performing across the five categories; on Left-Right Label–Slot Forms, it ties ReAct at the displayed precision (0.97). More importantly, these five types exhibit three distinct patterns.

(1) Repeated-Section Forms and Mixed-Slot-Layout Forms. The advantage of CRAFT is most pronounced on *Repeated-Section Forms* and *Mixed-Slot-Layout Forms*. On repeated-section forms, CRAFT achieves 0.59, clearly above the strongest baseline at 0.47. On mixed-slot-layout forms, the gap is even larger, with CRAFT reaching 0.70 while the best competing method remains at 0.55. These gains are consistent with the intended role of structure-aware validation and re-grounding: maintaining decisions across repeated units and recovering targets under noncanonical alignments. The aggregate subtype comparison does not isolate their individual effects.

(2) Mixed-Section Forms. CRAFT also remains the best-performing method on *Mixed-Section Forms*. The main challenge here is that a single form contains multiple local sections with dif-

Table 6: Representative case studies across both tracks. Outcomes are success (✓), partial success (▲), or failure (✗). Case metadata are listed below; L/R/U/D/ML denotes the original layout distribution.

Method	Instruction-Only 4	29	38	84	88	Multi-File 226	238	284
VLM+HTML+Image	✓	▲	✓	▲	✓	▲	▲	▲
VLM+HTML	▲	▲	✓	▲	✓	✗	▲	▲
VLM+Excel Skills+Coding	✗	✗	✗	✗	✗	✗	✗	✗
LangGraph	✓	▲	✓	▲	✓	✗	▲	▲
OpenAI Agents SDK	✓	▲	✓	✓	▲	✗	▲	▲
CrewAI	▲	✓	✓	✗	✓	✗	▲	▲
ReAct	✓	▲	✓	✓	✓	✗	▲	▲
SheetAgent	✗	✗	✗	✗	✗	✗	✗	✗
SheetBrain	✗	✗	✗	✗	✗	✗	✗	✗
SheetCopilot	✗	▲	▲	✗	✓	✗	✗	✗
SheetMind	✗	✗	✗	✗	✗	✗	✗	✗
CRAFT	✓	✓	✓	✓	✓	✓	✓	✓

Task ID	Case Type	Layout (L/R/U/D/ML)
4	Repeated-Section	0/6/2/0/2
29	Mixed-Slot-Layout	2/3/9/5/0
38	Left-Right Label–Slot	0/6/2/0/0
84	Bordered-Block	0/0/0/25/0
88	Mixed-Section	0/11/0/0/0
226	Bordered-Block	0/0/18/0/0
238	Mixed-Slot-Layout	0/0/0/17/0
284	Repeated-Section	0/20/0/0/0

ferent internal organizations, so the system cannot rely on one dominant filling pattern and must adapt its decisions across sections. Although the gap is smaller than in the two categories above, the result is consistent with the value of coordinating section-specific decisions.

(3) Bordered-Block Forms and Left-Right Label–Slot Forms. On *Bordered-Block Forms* and especially *Left-Right Label–Slot Forms*, the gap to strong baselines becomes smaller. These categories expose clearer and more regular structural cues, so even competing methods can recover much of the local grounding correctly, leading to relatively high accuracy overall. CRAFT remains competitive on these regular layouts, while its larger margins occur on mixed-slot and repeated-section forms.

Overall, these results suggest that the main advantage of CRAFT lies in handling structurally irregular forms, while still maintaining strong performance on more regular layouts.

Finding 1. ***(Larger Margins on Mixed and Repeated Layouts):*** *In this benchmark, CRAFT shows particularly clear gains on Mixed-Slot-Layout and Repeated-Section Forms. These subtype results describe where the system performs well; they do not separate layout effects from field count or source complexity.*

E.3 Case Study

In Table 6, we compare representative cases from both the ***Instruction-Only Track*** and the ***Multi-File Track***, and summarize each method's outcome as success, partial success, or failure. The selected cases highlight three recurring patterns. *First*, vanilla VLM baselines remain competitive on visually regular single-form cases, such as Case 38, where at least one rendering-based baseline achieves success. However, they become much less reliable on structurally irregular layouts: for example, in Case 29, which involves a mixed-slot-layout, the vanilla VLM baselines achieve only partial success. *Second*, spreadsheet-oriented agents show limited robustness beyond simpler spreadsheet-native settings. Although SheetCopilot (Li et al., 2023) succeeds in Case 88 and partially succeeds in Case 38, spreadsheet agents fail on most other structurally diverse cases, in-

cluding Cases 4, 29, and 84. *Third*, the *Multi-File Track* remains challenging for nearly all baselines. On Cases 226, 238, and 284, most competing methods drop to partial success or failure, indicating difficulty in consolidating evidence across files and grounding the retrieved values back to the target form. Across all selected cases, `CRAFT` is the only method that consistently achieves success.