

# A Benchmark and Framework for Evaluating Next Action Predictions in Spreadsheets

Tejas Agrawal<sup>1</sup> Vu Le<sup>2</sup> Sumit Gulwani<sup>2</sup> Gust Verbruggen<sup>3</sup>

## Abstract

Predictive code completion greatly accelerates how quickly developers work. In spreadsheets, despite being much more common, such auto-completion features are virtually non-existent. To address this gap, we introduce a benchmark for systems that observe a sequence of user actions in a spreadsheet and predict future actions. Two challenges are (1) the absence of edit histories in public spreadsheet corpora and (2) the complex space of spreadsheet actions (spatial, temporal, composite). To address (1), we manually curate 52 sequences of 12K actions that recreate spreadsheets from public corpora, seeded by parametrized heuristics and LLM refinement. To address (2), we propose an online evaluation that expects a prediction after each user action, accepts or rejects that prediction, updates the future actions upon acceptance, and repeats this until the target spreadsheet is obtained. We use multiple baseline predictors (including zero-shot LLMs, fine-tuned SLMs, and classical models) and analyze different properties that our benchmark teaches us, including but not limited to: properties of saved actions and false positives, efficiency, effect of user profiles, effect of triggers, and effect of context.

## 1. Introduction

Creating visually interesting spreadsheets requires typically hundreds of simple UI actions like selecting relevant ranges and clicking buttons to enter data or apply formatting. We introduce the first benchmarking dataset and framework for predictive auto-completion in spreadsheets, where the sys-

<sup>1</sup>Microsoft, Bangalore, India <sup>2</sup>Microsoft, Redmond, USA <sup>3</sup>Microsoft, Keerbergen, Belgium. Correspondence to: Tejas Agrawal <t-tagrawal@microsoft.com>, Gust Verbruggen <gverbruggen@microsoft.com>.

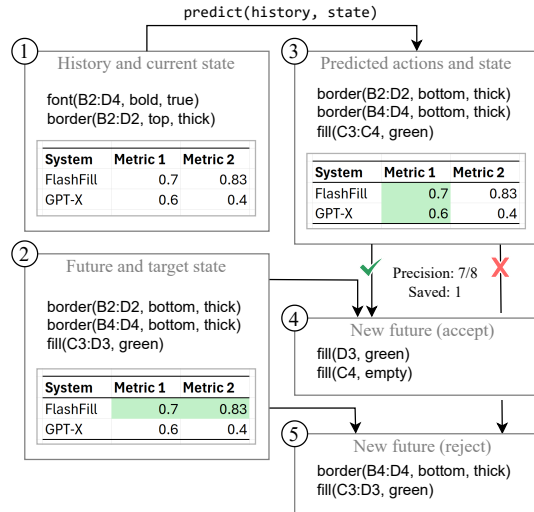


Figure 1. One step of the online evaluation on a synthetic example. ① The history (so far) produces the current state. ② The remaining future actions would produce the target state. ③ Given the history and current state, the system predicts three actions: two borders that match the future, and a fill that targets the wrong cell. At the (cell, property) level this gives precision 7/8; if accepted, the user would save one action overall (see below). The system then either ④ *accepts* or ⑤ *rejects* the prediction. On accept, the future is updated to remove operations the prediction already satisfied (the two borders) and a new fill(C4, empty) is prepended to undo the wrongly-coloured cell, leaving two actions instead of three. On reject, the future is unchanged and evaluation continues with the next user step.

tem observes the user performing a sequence of these low-level actions and then suggests the next actions to speed up the spreadsheet authoring process. This work thus bridges the gap between two emerging areas: code auto-completion and the rapid rise in developer productivity, and spreadsheet productivity and the rise of agents for spreadsheet authoring.

Code auto-completion has drastically evolved in recent years. Early symbolic systems would observe users making changes and use small programs to suggest repetitive edits to the rest of the file, like Blue-Pencil (Miltner et al., 2019). With the rise of transformers architectures, full lines of code could now be suggested from potentially noisy existing code (Svyatkovskiy et al., 2020). As models became more pow-

erful, so would their suggestions: modern coding assistants can suggest multiple functions in a pure auto-completion setting (Mastropaolo et al., 2023).

In spreadsheets, early assistants would auto-complete columns that could be derived from other columns, most notably FlashFill (Gulwani, 2011; Cambronero et al., 2023). Other assistants suggest formulas after the user explicitly signals the intent that they want a formula (Chen et al., 2021). Because of the complex interaction surface of spreadsheets, two dimensions of cells with stylistic properties, spreadsheet suggestions are constrained to very specific scenarios. More recently, the main focus for spreadsheet suggestions are agents that solve problems based on natural language intent (Ma et al., 2024; Li et al., 2023). For routine edits and repetitive patterns, the effort of invoking the assistive tool, formulating prompts, and waiting for responses exceeds the cost of direct manipulation, leading users to default to manual editing (Vaithilingam et al., 2022; Liang et al., 2025).

To the best of our knowledge, there is no work on generalized action suggestions for spreadsheets. One potential reason is that evaluating predictive assistants in spreadsheets is challenging for various reasons. First, as opposed to detailed version histories for code, there only exist very high-level corpora of spreadsheets as they evolve over time (Dou et al., 2016). Second, whereas each individual action (like coloring a cell or adding a border) is simple, different actions happen at (and affect) different parts of the spreadsheet in potentially many different orders, without a clear trigger of when to make a prediction. A potential solution is teacher-forced offline evaluation, where the system is always given the correct next state and each prediction is checked independently against a fixed target.

We address the first challenge by manually curating sequences of user actions from static workbooks, using a combination of heuristics and large language models for heuristic seeding to motivate diversity. Our dataset consists of 52 high-quality trajectories of workbooks being created from start to finish, each having between 35 and 821 steps with an average of 229 steps, for 11,907 steps in total.

We address the second challenge by proposing an online evaluation, instead of a static “given  $x$ , predict  $y$ ”. At each iteration, the system to be evaluated is given the past  $n$  actions and predicts a sequence of zero or more actions. Based on properties of the prediction, that prediction is either accepted or rejected. If accepted, the sequence of future actions is updated, for example, to remove successful actions and add actions that undo wrong moves. If rejected, the next user action is added to the history and the next iteration starts with  $n+1$  actions. This procedure is repeated until either there are no actions left to do (final spreadsheet obtained) or a threshold is reached. Our online evaluation has three advantages over a teacher-forced offline evaluation

at fixed time steps: (1) suggestions change the state that the model (and user) continues from, so errors can compound and later predictions depend on earlier ones; (2) the model cannot re-suggest the same easy actions and still appear accurate under stepwise scoring; and (3) the system is tested on revising or repairing its own prior suggestions.

*Example 1.* An example of a single iteration is shown in Figure 1. In this hypothetical prediction, the two border actions are correct and the fill is partially correct, for a total (per-cell) precision of 7/8. Because one correction operation is added to the new future to remove the wrong fill, the user saves one action.

Our framework then computes different metrics at different levels (per simulation, per prediction, per action) that allow targeted improvement of action prediction systems. We use multiple solvers as baseline prediction systems (such as zero-shot LLMs, finetuned SLMs, and some classical machine learning models) to highlight what we can learn using these metrics. For example, more powerful models save more actions than weaker models (33% on GPT-5 with reasoning versus 18% on GPT-5 mini) and fine-tuned 360M-parameter models match GPT-5 (both at 27%). This indicates that the task is learnable. Low-precision acceptance heuristics yield negative savings (−19%), confirming that abstention based on net user benefit is crucial. Being able to predict at each action saves more actions (27%) than after every four actions (17%), indicating that good triggers or cheaper models are worth investigating. Section 5 describes many more insights.

In summary, we make the following contributions

- **Benchmark dataset.** We curate a dataset of 52 spreadsheet creation trajectories totaling 11,907 operations, each one validated by humans.
- **Online evaluation.** We propose an evaluation framework that evaluates systems end-to-end by modeling user acceptance behavior, dynamically adapting ground-truth trajectories after predictions, and computing metrics that capture real-world utility.
- **Design insights.** We demonstrate our dataset and framework using baselines to show how these metrics allow targeted improvement of predictive assistants for spreadsheets.

and release artifacts at [www.github.com/Tej-55/NAPE](https://www.github.com/Tej-55/NAPE).

## 2. Benchmarks

Each benchmark consists of a sequence of parametrized actions that a user takes to build a final workbook from scratch. Table 1 lists the supported actions.

Operation	Explanation
input(d, v)	Enter v (value or matrix) at location d.
merge(d)	Merge range d.
format(d, f)	Set number format f at range d.
fill(d, c)	Set background fill color c at range d.
font(d, p, v)	Set font property $p \in \{\text{bold, italic, size, color, underline, name}\}$ to v at d.
border(d, s, v)	Set border on side $s \in \{\text{left, right, top, bottom, outside, all}\}$ to style v at d.
align(d, t, v)	Set text property $t \in \{\text{horizontal, vertical, orientation, wrap}\}$ to v at d.
paste(d, s, m)	Paste $m \in \{\text{style, value, both}\}$ from s to d.
autofill(d, s)	Auto-fill range d from source s.

Table 1. Different user actions available in spreadsheet software.

We reconstruct user trajectories from static, publicly available workbooks (Singh et al., 2023) through a three-stage pipeline: (1) symbolic cold-start generation, (2) LLM-assisted refinement, and (3) human annotation. The first two stages reduce human annotation effort while introducing natural variation in construction sequences, providing annotators with an initial trajectory to work from rather than requiring them to author sequences from scratch.

**Symbolic cold-start.** Given a target workbook, we decompose the final state into cell-level operations and merge adjacent identical operations into range actions (e.g., font(A1, bold, true) + font(A2, bold, true) become font(A1:A2, bold, true)). To introduce variability, we sample from a space of preference settings that control merging strategy, region ordering, and formatting order. A vision-language model annotates each sheet with semantic metadata (regions, dependencies, pasted ranges). Full details of this pipeline appear in Appendix A.

**LLM refinement.** An LLM-based judge-editor loop serves as a helper step, identifying and correcting obvious unnatural patterns in the symbolic sequences: consolidating scattered cell-by-cell formatting into range operations, removing stray formatting on unused areas, and adjusting sequences to follow more natural patterns. This step reduces the annotation burden by handling mechanical corrections and all revisions are validated to ensure they reach the target workbook state.

**Human annotation.** Human annotators perform a final pass by correcting remaining unnatural subsequences. As evidence of the substantial restructuring performed by annotators, the mean normalized edit distance between pre-annotation and final trajectories is 0.69 (median 0.77), and 19 of 52 trajectories were effectively rewritten from scratch (distance > 0.8); see Figure 11 in the Appendix. Common refinements include grouping content with its formatting, reordering to establish structure before detail, breaking bulk

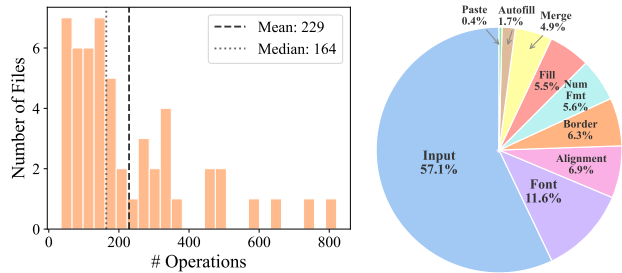


Figure 2. Dataset characteristics. (left) Distribution of file sizes by number of operations. (right) Composition of operations by category, showing diversity of formatting and content operations.

inputs into individual cell entries, and trimming ranges to content boundaries.

**Dataset statistics.** Our final dataset comprises 52 sheets with a combined 11,907 operations. Sequence lengths range from 35 to 821 (mean 229, median 164). Only one sheet is used per workbook to maximise diversity. Figure 2 shows the size distribution and operation breakdown.

### 2.1. Empirical Predictability

Not all actions in a trajectory are predictable from context alone. To estimate an empirical upper bound on predictability, we conduct an oracle experiment: for each trajectory, we invoke frontier reasoning models at every step with the full operation history and the corresponding sheet image at that time, and ask it to predict zero or more subsequent actions. Each model generates two generations per step, and we repeat this process across four LLMs: Opus 4.6, Opus 4.7, GPT-5.4, and GPT-5.5, all with high reasoning effort. We then take the union of all correctly predicted (cell, property) pairs across invocations.

Across the 52 trajectories, 68% of all ground-truth properties appear in this union, with a per-trajectory median of 66.3% (44 of 52 trajectories exceed 50%). This establishes that the vast majority of spreadsheet actions are, in principle, predictable from editing history, and sets a theoretical ceiling for online evaluation performance.

## 3. Online Evaluation

Online evaluation judges the system under the same conditions a real user operates in, where each accepted suggestion affects the future actions that a user still has to perform. To this end, we expose a function that evaluates a given assistant using an online evaluation loop, which performs an on-policy rollout over worksheet states. A high-level overview of this evaluation, which ignores some evaluation settings and logging metrics, is shown in Figure 4. Some acceptance criteria are explored in our experiments (Section 5).

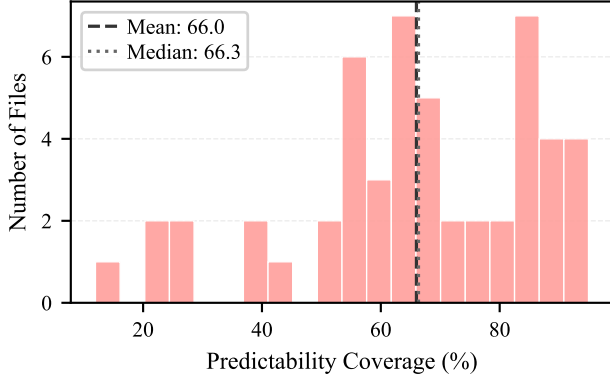


Figure 3. Distribution of predictability coverage across the 52 benchmark trajectories.

The `update` step adapts the remaining future after an accepted prediction. It removes operations whose effects are already satisfied, prepends inverses for any false positives, and verifies that the adapted sequence still reaches the target state (like the partial fill in Figure 1). A final `patch` step simulates the adapted future and synthesizes corrections for any residual discrepancies, ensuring the updated sequence always produces the original target state.

### 3.1. Metrics

We define metrics at three levels of granularity.

#### 3.1.1. PROPERTY/ACTION LEVEL

At the finest granularity, we compare individual (cell, property) pairs between the predicted state and the target state, considering only explicitly modified properties. Each pair is classified as follows. **True positive (TP)**: the property appears in both states with the same value; the prediction correctly anticipates the user’s edit. **False positive (FP)**: the property is present in the prediction but absent in the target, introducing an unwanted edit the user must undo. **False negative (FN)**: the property is missing from the prediction but required by the target, leaving work the user must perform manually. **Mismatch (MM)**: the property appears in both states but with different values, producing an incorrect edit that the user must correct.

#### 3.1.2. PREDICTION LEVEL

A prediction refers to all actions proposed in one iteration. These metrics can be used by `accept`. **Precision**: fraction of predicted properties that are correct,  $TP/(TP + FP + MM)$ , indicating how few errors a prediction introduces. **User actions saved (UAS)**: the reduction in user effort if the prediction were accepted, computed by simulating acceptance and measuring how much the future diverges; positive values help the user, negative values hurt.

```

1 class ISolver:
2   def predict(s: Spreadsheet,
3             h: Action[]) → Action[]:
4     // to implement
5
6   def evaluate(ISolver solver,
7               Action[] sequence,
8               bool repredict = False):
9     S ← Spreadsheet.empty()
10    H ← []
11    F ← sequence
12    while F:
13      while True:
14        P ← solver.predict(S, H)
15        if not accept(P, H, F):
16          break
17        S ← execute(P, S)
18        F ← update(F, P, S)
19        H ← H + P
20        if not repredict: break
21
22  def update(F, P, S):
23    Sp ← execute(P, S)
24    Sf ← execute(F, Sp)
25    F' ← []
26    // remove satisfied operations and only keep
27    for op in F:
28      if not satisfied(op, Sp):
29        F' += residual(op, Sp)
30    // undo false positives with inverse operations
31    for p in false_pos(Sp, Sf):
32      F' ← [invert(p)] + F'
33    // verify and patch
34    if execute(F', Sp) ≠ Sf:
35      F' ← patch(F', Sp, Sf)
36    return F'

```

Figure 4. Pseudo-code for the interface that assistants should implement and the exposed `evaluate` function that performs online evaluation. Computing metrics and additional hyperparameters are omitted for brevity.

#### 3.1.3. EMULATION LEVEL

These metrics capture system performance over a full trajectory. **User actions saved (UAS)**: percentage reduction in user effort,  $(L_{\text{initial}} - L_{\text{final}})/L_{\text{initial}}$ . This is our primary metric for performance. **Acceptance rate (AR)**: fraction of predictions accepted. **Average precision (PREC)**: macro-average precision across all predictions. **Predictability Coverage (PCOV)**: fraction of *predictable* (cell, property) pairs that the system actually produced, where the predictable set is the oracle union from §2.1. This normalizes by what is learnable from context, giving a ceiling-relative measure rather than one diluted by inherently unpredictable edits.

## 4. Experimental Setup

### 4.1. Solvers

We evaluate three families of solvers for our baselines.

**Zero-shot LLMs.** Predictions are made by representing actions as text, where `fill(A1, green)` is represented as `FILL | A1 | green`. The prompt instructs the model to suggest next actions in a zero-shot manner and, besides instructions,

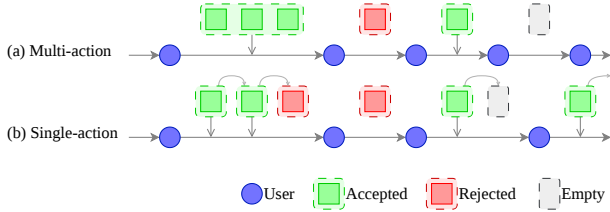


Figure 5. Two prediction settings. (a) *Single-action* ( $k = 1$ , re-predict): the solver is queried for one op at a time; after each acceptance the solver is re-queried before the next user step. (b) *Multi-action* ( $k \geq 1$ ): the solver may emit a block of up to  $k$  ops per query, and the cursor advances by the length of any accepted block.

includes: (1) the recent operation history, and (2) the valid syntax for available operations. We evaluate GPT-5-R, GPT-5, GPT-5-R mini, and GPT-5 mini, where the “-R” suffix denotes models run in reasoning mode (set to low); models without the suffix are run in non-reasoning chat mode. Unless specified otherwise, the default model is GPT-5 (with no reasoning). The full prompt template is provided in the Appendix D.

**Fine-tuned SLMs.** We fine-tune SmolLM2 models (135M and 360M parameters) on synthetically generated operation sequences. Training data is produced using the same symbolic pipeline described in Section 2, applied to a separate pool of 12K workbooks disjoint from the evaluation set. Each training example consists of a full operation sequence for a single sheet trajectory—averaging 163 operations—identical in format to the evaluation data. We window each trajectory using width 32 and stride 16, yielding  $\pm 45K$  data points for supervised fine-tuning.

**Classical ML.** We use four types of non-generative predictors on operation sequences: trained  $n$ -gram, online  $n$ -gram, LSTM, and XGBoost. The trained  $n$ -gram, LSTM, and XGBoost models are trained on the same synthetic data as the SLMs. The online  $n$ -gram requires no training and learns patterns from past actions within each trajectory. Each predicts the next operation from a sliding window of recent actions using relative features (offsets from the previous action) rather than absolute features (cell addresses). We found this to generalize better across trajectories in preliminary experiments. Full architecture details are in Appendix B.

## 4.2. Evaluation Settings

We use two evaluation settings that differ in how many actions the solver produces per invocation.

**Multi-action prediction.** The solver is invoked once on the current history and predicts a variable-length sequence of actions in a single shot. This setting tests both prediction quality and the solver’s ability to decide when to stop:

longer correct predictions save more user actions, but stopping too late introduces false positives that the user must reject or undo. This is a natural setting for zero-shot LLMs, which can produce arbitrary-length completions and decide internally when to stop.

**Single-action re-predict.** The solver predicts exactly one action per invocation. If accepted, it is immediately re-invoked on the updated history to predict the subsequent action; this continues until a prediction is rejected. We use this setting to evaluate solvers that (for now) lack the capacity to autonomously decide when to stop predicting, such as fine-tuned SLMs and classical sequence models. It removes the stopping burden entirely: the evaluation loop controls termination via the acceptance heuristic. This is analogous to a user accepting word-by-word completions (repeatedly pressing *tab* in a code editor).

## 4.3. Hyperparameters

We vary four key hyperparameters that control the evaluation. **Prediction stride** ( $s$ ): the number of user actions between successive solver invocations;  $s=1$  invokes the solver after every action. **Context window** ( $c$ ): the number of recent operations provided to the solver as history. **Max actions per prediction** ( $m$ ): the number of actions the solver may produce per invocation;  $m=\infty$  for multi-action,  $m = 1$  for single-action re-predict. **Context shortening**: whether long cell values in the history are truncated to a fixed character limit before being sent to the solver, reducing token usage. Unless stated otherwise, we use  $s = 1$ ,  $c = 32$ ,  $m = \infty$  (multi-action) or  $m = 1$  (single-action re-predict), and context shortening enabled.

## 4.4. Acceptance Heuristics

The acceptance heuristic (`accept` in Figure 4) determines whether a prediction is applied or rejected. We experiment with several heuristics that combine precision and savings constraints, as shown in Table 5.

The GREEDY heuristic models a user who accepts any suggestion that advances their goal, regardless of minor errors. Precision-only heuristics (P90, P60) model a user who accepts predictions that look mostly correct, even if they do not reduce overall effort. The ALWAYS baseline accepts every prediction regardless of quality, serving as an upper bound on acceptance rate and a stress test for the ground-truth adaptation procedure.

## 5. Experiments

**The task is learnable.** We observe a correlation between model capability and prediction performance (Tables 4 and 2). The consistent gradient from GPT-5 mini (20.1%

## Evaluating Next Action Prediction Systems for Spreadsheets

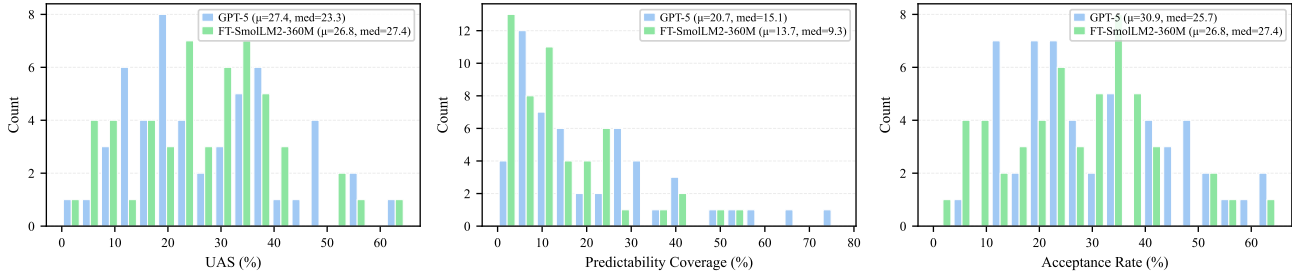


Figure 6. Per-file distributions across the 52 benchmark trajectories, comparing GPT-5 (blue) and FT-SmolLM2-360M (green), both run with single-action repredict and default settings. (Left) User Actions Saved. (Center) Predictability Coverage, the fraction of properties an oracle deemed predictable that the model correctly produced. (Right) Acceptance Rate. Legend entries show mean ( $\mu$ ) and median for each model.

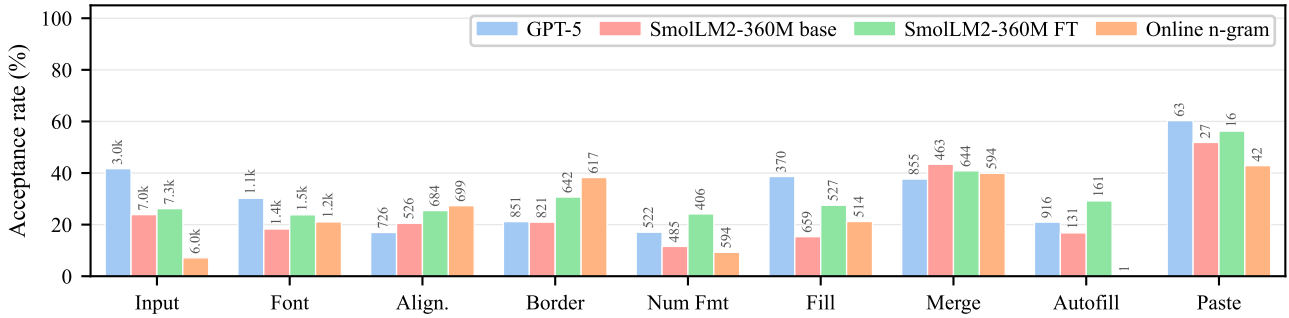


Figure 7. Per-category acceptance rate across four solvers. Numbers above each bar give the count of predictions that solver made in that category.

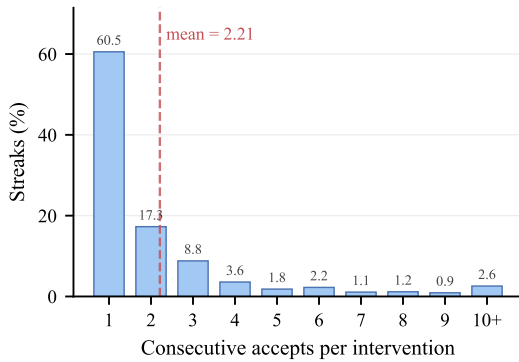


Figure 8. Distribution of acceptance streak lengths for GPT-5 in the single-action repredict setting (1,204 streaks across 52 trajectories). Most streaks are short, but a heavy tail of longer runs (3+ consecutive accepts) drives most of the realised user-action savings.

UAS multi-action, 18.0% single-action) to GPT-5-R (26.6% multi-action, 32.7% single-action) confirms that action prediction is a learnable task, not lucky guessing. Stronger models systematically perform better, suggesting significant room for improvement through specialized training or architectures optimized for this domain. Fine-tuning further supports this: SmoLM2-360M improves from 21.7% to

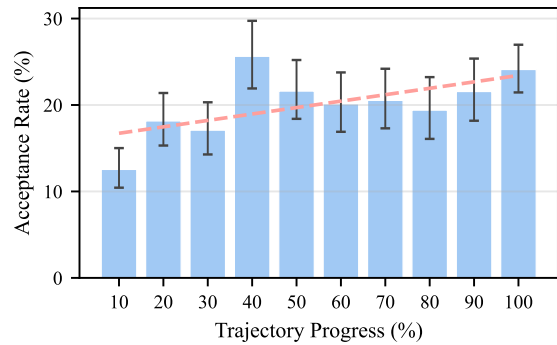


Figure 9. Acceptance rate by trajectory progress (10% bins). As users progress through their spreadsheet editing task, acceptance rates increase, suggesting predictions become more accurate as more context becomes available. (uses default settings)

26.8% UAS and SmoLM2-135M from 18.3% to 23.2% after fine-tuning on operation sequences (Table 2), narrowing the gap to GPT-5 (27.4%) despite their much smaller size.

**Abstention is key.** The ALWAYS heuristic, which accepts every non-empty prediction, yields strongly negative savings ( $-19.2\%$  UAS) at only 9.3% precision (Table 5); in fact, 51 of 52 trajectories hit the enforced user-step cap of 120% of ground truth, so the unconstrained number would

## Evaluating Next Action Prediction Systems for Spreadsheets

MODEL	UAS $\uparrow$	AR $\uparrow$	PREC $\uparrow$	PCOV $\uparrow$
<i>Zero-shot LLMs</i>				
GPT-5-R	32.7	29.4	41.6	24.8
GPT-5-R mini	28.2	25.5	37.0	20.9
GPT-5	27.4	30.9	44.8	20.7
GPT-5 mini	18.0	16.8	21.9	10.7
<i>SLMs (base)</i>				
SmolLM2-360M	21.7	22.3	29.7	9.6
SmolLM2-135M	18.3	19.0	24.8	8.9
<i>SLMs (finetuned)</i>				
FT-SmolLM2-360M	26.8	26.8	33.7	13.7
FT-SmolLM2-135M	23.2	23.1	30.6	13.0
<i>Classical sequence models</i>				
LSTM	5.7	5.5	12.4	2.4
Trained $n$ -gram	3.8	3.9	11.9	0.7
XGBoost	2.9	2.3	6.5	1.0
Online $n$ -gram	12.0	14.7	20.4	11.1

Table 2. Model comparison in single-action repredict ( $s=1$ ,  $c=32$ , GREEDY). -R means reasoning (low).

PARAM	VALUE	UAS $\uparrow$	AR $\uparrow$	PREC $\uparrow$	PCOV $\uparrow$
Stride ( $s$ )	1*	27.4	30.9	44.8	20.7
	2	22.6	36.5	48.4	15.9
	4	16.8	42.3	53.2	9.4
	8	10.6	43.7	55.1	7.1
Context ( $c$ )	8	19.9	24.1	39.7	13.6
	32*	27.4	30.9	44.8	20.7
	128	30.0	32.5	47.8	28.5
	512	30.8	33.7	47.9	31.0
Shortening	on*	27.4	30.9	44.8	20.7
	off	27.7	31.2	44.3	21.4
Repredict	on*	27.4	30.9	44.8	20.7
	off	20.3	30.6	44.1	15.2

Table 3. Hyperparameter ablation (single-action repredict, GPT-5, GREEDY). Defaults marked \*.

be far worse. This illustrates that LLMs do not natively know when to abstain (no intent is available) or when to stop predicting, two key areas to focus on for future predictors. Since smaller models and classical sequence solvers cannot be fairly evaluated in the multi-action setting, we additionally report results in the single-action repredict setting (Table 2), an equivalent setup that levels the playing field by restricting every solver to one action per call. Under this protocol, SmolLM2-360M FT (26.8% UAS) approaches GPT-5 (27.4%) and even the training-free ONLINE  $n$ -GRAM (12.0%) becomes a non-trivial baseline, highlighting the potential of smaller models and motivating research on explicit abstention objectives rather than on raw generative capacity alone.

**Cheaper predictions or triggers.** The stride  $s$  dramatically affects system behavior (Table 6). At  $s=1$ , the system

MODEL	UAS $\uparrow$	AR $\uparrow$	PREC $\uparrow$	PCOV $\uparrow$
GPT-5-R	26.6	13.3	27.8	30.7
GPT-5	22.3	20.0	36.0	19.6
GPT-5-R mini	21.3	9.6	23.2	24.3
GPT-5 mini	20.1	10.1	19.0	17.3

Table 4. Multi-action prediction ( $s=1$ ,  $c=32$ , GREEDY).

HEUR.	RULE	UAS $\uparrow$	AR $\uparrow$	PREC $\uparrow$	PCOV $\uparrow$
GREEDY	$\ell \geq 1$	22.3	20.0	36.0	19.6
HYBRID-1	$p \geq .9, \ell \geq 1$	21.8	16.6	39.4	17.3
GREEDY-2	$\ell \geq 2$	20.3	10.5	37.6	15.9
HYBRID-2	$p=1, \ell \geq 2$	17.5	7.9	40.1	10.7
P100	$p=1$	19.9	21.8	38.2	20.1
P90	$p \geq .9$	17.0	23.3	36.0	25.5
P60	$p \geq .6$	13.3	26.9	32.9	33.0
ALWAYS	—	-19.2	100.0	9.3	8.1

Table 5. Effect of acceptance heuristics (multi-action, GPT-5,  $s=1$ ,  $c=32$ ). Rule:  $p$  = precision,  $\ell$  = steps saved.

attempts a prediction after every user action, achieving the highest UAS (22.3%) despite the lowest acceptance rate (20.0%). At  $s=8$ , the acceptance rate climbs to 36.5% but UAS collapse to 9.8%. The same pattern holds in the single-action setting (Table 3):  $s=1 \rightarrow 8$  moves AR from 30.9% to 43.7% while UAS falls from 27.4% to 10.6%. A cheap trigger that can be invoked at every iteration is therefore a key focus area to improve overall utility.

**Easier and harder actions.** Per-category acceptance rates (Figure 7) split cleanly into two groups across all solvers: content-heavy operations (`input`, `paste`, `fill`, `merge`) are accepted at noticeably higher rates than presentational ones (`align`, `number format`, `border`). GPT-5 dominates on `input`, reflecting its strong language prior on cell content, while the training-free online  $n$ -gram holds up well on structural categories driven by local repetition (`merge`, `border`, `paste`) but collapses on semantic categories such as `input`.

Comparing the base and fine-tuned SmolLM2-360M makes the effect of in-domain training concrete: the largest accuracy gains land on the categories the base handled worst—`number format`, `fill`, `border`, and `autofill` all jump by roughly 10 points—while content categories shift only marginally. At the same time, the fine-tuned model issues fewer `border` and `fill` predictions and more `merge`, `align`, and `autofill` ones, suggesting it learns both *when* to predict each category and *how* to predict it more reliably.

**Acceptance streaks.** We define an *acceptance streak* as a maximal run of consecutive accepted single-action predictions between two user interventions (Figure 8). The distribution for our base run is heavy-tailed: most streaks

PARAM	VALUE	UAS $\uparrow$	AR $\uparrow$	PREC $\uparrow$	PCOV $\uparrow$
Stride ( $s$ )	1*	22.3	20.0	36.0	19.6
	2	19.5	24.4	39.4	16.0
	4	14.7	33.4	45.1	12.2
	8	9.8	36.5	49.4	6.8
Context ( $c$ )	8	16.2	17.5	33.8	12.3
	32*	22.3	20.0	36.0	19.6
	128	27.6	19.7	37.3	30.0
	512	26.2	19.0	35.9	32.0
Shortening	2048	27.4	19.4	36.6	33.0
	on*	22.3	20.0	36.0	19.6
	off	24.2	21.5	38.4	21.4
Num ops ( $m$ )	1	20.3	30.6	44.1	15.2
	4	20.8	23.1	39.2	19.4
	8	22.1	19.4	36.1	18.9
	16	21.8	20.6	36.2	20.2
	$\infty^*$	22.3	20.0	36.0	19.6

Table 6. Hyperparameter ablation (multi-action, GPT-5, GREEDY). Defaults marked \*.

are short (median 1), yet the mean of 2.21 implies that each *successful* intervention commits roughly twice as many operations as the user would otherwise type, and 5% of streaks chain seven or more accepts. The other solvers we evaluated in Table 2 exhibit the same qualitative pattern, with mean streak lengths clustered in the 1.6–2.3 range. This burst-like behaviour reconciles the seemingly modest per-prediction acceptance rate ( $\sim 32\%$ ) with the substantially larger reported UAS: predictors do not save effort uniformly, but rather lock onto locally repetitive structure (row fills, border propagation, repeated formatting) and chain several confident predictions before the user must reassert control.

**Conservative acceptance is not strictly better.** The GREEDY-2 heuristic demands higher utility than GREEDY, and HYBRID-2 layers an additional perfect-precision constraint on top (Table 5). As expected, acceptance rates fall (20.0%  $\rightarrow$  10.5%  $\rightarrow$  7.9%), but UAS also falls (22.3%  $\rightarrow$  20.3%  $\rightarrow$  17.5%). Opportunistic acceptance (GREEDY) yields the best cumulative benefit, while strict filters discard borderline-but-useful predictions faster than they remove harmful ones.

**Precision alone does not ensure utility.** Even at p90, which accepts any prediction with  $\geq 90\%$  precision regardless of utility, the system delivers only 17.0% UAS – 4.8 points below HYBRID-1 (21.8%) and 5.3 points below GREEDY (22.3%) (Table 5). P100 (perfect precision) does better at 19.9% but is still surpassed by HYBRID-1. Precision measures correctness of *individual* predicted properties, not whether a prediction actually advances the user toward completion; a heuristic that ignores utility leaves savings on the table.

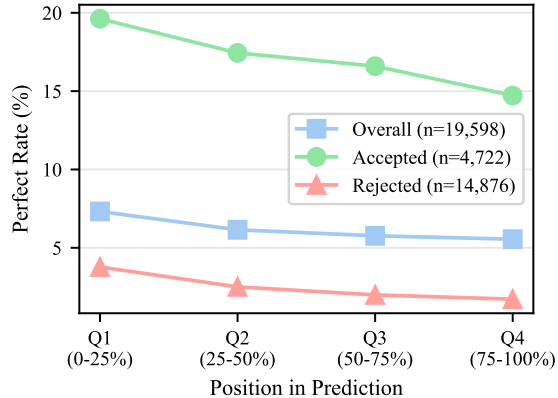


Figure 10. Perfect rate (precision = 100%) by operation position within predictions (uses default settings). Operations are grouped by their normalized position (quartile) within each prediction. Accepted predictions show consistently higher perfect rates across all positions compared to rejected predictions.

**The right context length.** Context length has a clear effect on performance, but with quickly diminishing returns (Tables 6 and 3). In the multi-action setting, increasing the window from 32 to 128 prior operations lifts UAS from 22.3% to 27.6%, while leaving the acceptance rate essentially unchanged. Gains taper rapidly beyond this point: 512 and 2048 contexts yield 26.2% and 27.4% UAS respectively, indistinguishable from 128. The single-action setting shows the same saturation (27.4% at  $c=32$ , 30.0% at 128, 30.8% at 512). This suggests that while long-range history helps, most of the predictive signal resides within the most recent  $\sim 128$  operations.

**The right prediction length.** Unlimited prediction scope performs best in the multi-action setting; constraining the per-call budget never improves UAS (Table 6). Capping the model at  $m=1$  inflates the acceptance rate to 30.6% but yields only 20.3% UAS, because each accepted prediction now saves at most one step. Mid-range caps ( $m=4$ ,  $m=8$ ,  $m=16$ ) all hover around 21–22% UAS, roughly growing lightly with the limit. This suggests the model already self-regulates prediction length based on confidence, and external limits prevent it from exploiting high-confidence opportunities. Notably, the  $m=16$  setting performs worse than  $m=8$  on UAS, this could be due to higher caps encouraging the model to “fill” its budget with lower-confidence edits rather than stopping naturally.

**Acceptance rate increases as trajectories progress.** The acceptance rate rises with trajectory progress, from  $\sim 12.5\%$  in the first 10% of a trajectory to  $\sim 24\%$  by the end (Figure 9). This indicates a cold-start problem: without established patterns to recognize, early predictions are largely guesses. As context grows, the model identifies repetitive formatting,

table structures, and user-specific conventions that make later predictions more reliable. This suggests that adaptive triggering—predicting less frequently early in trajectories and more frequently later—could improve overall efficiency by avoiding the high rejection rate of early predictions.

**Overpredictions.** Figure 10 shows how the perfect-precision rate of an action varies with its position inside the prediction. For *accepted* predictions, the perfect rate decays monotonically from  $\sim 19.5\%$  in the first quartile to  $\sim 14.7\%$  in the last: when the assistant knows what to do, it starts by doing the right thing and then drifts. *Rejected* predictions are subpar throughout and follow the opposite, slightly declining trend ( $\sim 3.5\% \rightarrow \sim 1.5\%$ ). This suggests that abstention is relevant at two granularities: at the prediction level (to suppress entirely-bad predictions) and at the action level within an otherwise good prediction (to truncate before the tail degrades). The latter is reinforced by our observation that a substantial share of non-empty predictions contribute at least one true positive to the final state yet are still rejected—a gap that represents real improvement potential, since many predictions contain useful content but are discarded because of accompanying errors that violate acceptance heuristics.

## 6. Related Work

**Spreadsheet automation.** Prior work on spreadsheet automation spans two paradigms. *Natural-language-to-action* systems translate explicit user commands into operations: SheetCopilot (Li et al., 2023) and SpreadsheetBench (Ma et al., 2024) benchmark this approach, while TableTalk (Liang et al., 2025) studies how users interact with such assistants and finds that for routine edits, the overhead of formulating prompts exceeds the cost of direct manipulation. *Action-to-action* systems instead suggest completions from prior user actions without explicit invocation: FlashFill (Gulwani, 2011) synthesises string transformations from input-output examples, FlashRelate (Barowy et al., 2015) extracts relational data from semi-structured spreadsheets, and SpreadsheetCoder (Chen et al., 2021) predicts formulas from tabular context. Our benchmark targets a third, underexplored category: *modeless* action suggestion, where the system continuously observes user edits and proactively suggests next actions without being invoked.

**Modeless edit suggestion and code completion.** Modeless suggestion systems, which surface predictions without explicit user invocation, are well established in code editing. Early symbolic systems like Blue-Pencil (Miltner et al., 2019) suggested repetitive edits, IntelliCode (Svyatkovskiy et al., 2020) extended this to full-line completion, and modern assistants like GitHub Copilot can suggest multi-line blocks (Mastrolopaolo et al., 2023). Izadi et al. (2024) provide

a practical evaluation of language models for code completion, establishing methodologies that inform our evaluation design. Mozannar et al. (2024) study when to surface a suggestion in AI-assisted programming, directly motivating our investigation of prediction stride and acceptance heuristics. In GUI automation, ASSISTGUI (Gao et al., 2024) benchmarks task-oriented desktop automation, a setting that shares the sequential, state-modifying nature of spreadsheet editing. Most directly related, Shaikh et al. (2026) learn next-action predictors from human-computer interaction traces in desktop UI settings; our work addresses the analogous problem in spreadsheets.

**Interactive evaluation.** Evaluating interactive assistants under realistic conditions is a cross-domain challenge. In information retrieval, Borlund (2009) argues for user-centred evaluation that assesses systems as they would actually be used. In recommender systems, Castells & Moffat (2022) show that offline metrics diverge from actual user utility due to the unobserved-truth problem, and Ferrari Dacrema et al. (2022) demonstrate that offline evaluation of interacting recommendations fails to capture their combined impact. Offline policy evaluation in sequential settings (Mandel et al., 2014) faces analogous distribution shift. Spreadsheet assistance exhibits similar issues: predictions reshape state, users vary in which suggestions they accept, and the same final workbook can result from many different action sequences. Our online evaluation loop addresses these challenges by simulating acceptance decisions and dynamically adapting the ground truth.

## 7. Conclusion

We introduce the first benchmark for the next action prediction task in spreadsheets. Through synthetic generation and refinement, we create 52 benchmark tasks that span 12K actions in total. Online evaluation allows us to evaluate how users perceive using the predictor. Our experiments demonstrate that this benchmark can be used to gain actionable insights for predictors, such as room for improvement by abstention, cheap triggers, stopping criteria, and more.

## Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here. Note that, whereas LLMs are used in our baselines to demonstrate the benchmark, we specifically encourage research on less energy intensive methods to solve this problem.

## References

- Barowy, D. W., Gulwani, S., Hart, T., and Zorn, B. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. *ACM SIGPLAN Notices*, 50(6):218–228, 2015.
- Borlund, P. User-centred evaluation of information retrieval systems. *Information Retrieval: searching in the 21st Century*, pp. 21–37, 2009.
- Cambronero, J., Gulwani, S., Le, V., Perelman, D., Radhakrishna, A., Simon, C., and Tiwari, A. Flashfill++: Scaling programming by example by cutting to the chase. *Proceedings of the ACM on Programming Languages*, 7 (POPL):952–981, 2023.
- Castells, P. and Moffat, A. Offline recommender system evaluation: Challenges and new directions. *AI magazine*, 43(2):225–238, 2022.
- Chen, X., Maniatis, P., Singh, R., Sutton, C., Dai, H., Lin, M., and Zhou, D. Spreadsheetscoder: Formula prediction from semi-structured context. In *International Conference on Machine Learning*, pp. 1661–1672. PMLR, 2021.
- Dou, W., Xu, L., Cheung, S.-C., Gao, C., Wei, J., and Huang, T. Venron: a versioned spreadsheet corpus and related evolution analysis. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pp. 162–171, 2016.
- Ferrari Dacrema, M., Felicioni, N., and Cremonesi, P. Offline evaluation of recommender systems in a user interface with multiple carousels. *Frontiers in big Data*, 5: 910030, 2022.
- Gao, D., Ji, L., Bai, Z., Ouyang, M., Li, P., Mao, D., Wu, Q., Zhang, W., Wang, P., Guo, X., et al. Assistgui: Task-oriented pc graphical user interface automation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13289–13298, 2024.
- Gulwani, S. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46 (1):317–330, 2011.
- Izadi, M., Katzy, J., Van Dam, T., Otten, M., Popescu, R. M., and Van Deursen, A. Language models for code completion: A practical evaluation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.
- Li, H., Su, J., Chen, Y., Li, Q., and Zhang, Z.-X. Sheetcopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems*, 36:4952–4984, 2023.
- Liang, J. T., Kumar, A., Bajpai, Y., Gulwani, S., Le, V., Parnin, C., Radhakrishna, A., Tiwari, A., Murphy-Hill, E., and Soares, G. Tabletalk: Scaffolding spreadsheet development with a language agent. *ACM Transactions on Computer-Human Interaction*, 32(6):1–49, 2025.
- Ma, Z., Zhang, B., Zhang, J., Yu, J., Zhang, X., Zhang, X., Luo, S., Wang, X., and Tang, J. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37: 94871–94908, 2024.
- Mandel, T., Liu, Y.-E., Levine, S., Brunskill, E., and Popovic, Z. Offline policy evaluation across representations with applications to educational games. In *AAMAS*, volume 1077, 2014.
- Mastro Paolo, A., Pascarella, L., Guglielmi, E., Ciniselli, M., Scalabrino, S., Oliveto, R., and Bavota, G. On the robustness of code generation techniques: An empirical study on github copilot. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 2149–2160. IEEE, 2023.
- Miltner, A., Gulwani, S., Le, V., Leung, A., Radhakrishna, A., Soares, G., Tiwari, A., and Udupa, A. On the fly synthesis of edit suggestions. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–29, 2019.
- Mozannar, H., Bansal, G., Fourney, A., and Horvitz, E. When to show a suggestion? integrating human feedback in ai-assisted programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 10137–10144, 2024.
- Shaikh, O., Teutschbein, V., Gandhi, K., Chi, Y., Haber, N., Robinson, T., Ram, N., Reeves, B., Yang, S., Bernstein, M. S., et al. Learning next action predictors from human-computer interaction. *arXiv preprint arXiv:2603.05923*, 2026.
- Singh, M., Sanchez, J. C., Gulwani, S., Le, V., Negreanu, C., and Verbruggen, G. Cornet: Learning spreadsheet formatting rules by example. *Proceedings of the VLDB Endowment*, 16(12):4058–4061, 2023.
- Svyatkovskiy, A., Deng, S. K., Fu, S., and Sundaresan, N. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 1433–1443, 2020.
- Vaithilingam, P., Zhang, T., and Glassman, E. L. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*, pp. 1–7, 2022.

## A. Dataset Construction Details

We leverage a combination of symbolic heuristics, LLM refinement and human refinement to reconstruct user trajectories from static workbooks. First, we leverage an LLM to annotate spreadsheets with semantic metadata, such as table and header information. Second, we use symbolic heuristics to generate low-level steps, like inserting values or applying formatting. Third, an LLM is used to make the heuristic sequences more natural by reordering and combining actions. Fourth, we perform a manual inspection over these trajectories and resolve last-mile inconsistencies to ensure high quality data. The following four subsections describe these in detail.

### A.1. Worksheet Annotation

Users do not create entire spreadsheets value by value. Data is often entered from external sources (other workbook or web pages) and repetitive parts are created by copy and pasting a related part of the sheet and then adapting it. As these relations are hard to determine symbolically, we use a vision-language model (VLM) to annotate each spreadsheet with four types of metadata.

1. **Regions** are distinct areas of the spreadsheet that have some logical meaning, such as tables (including surrounding headers and labels), title blocks, or calculation areas. We imagine that a user typically finishes each region before moving to the next. For each region, the VLM predicts:
  - A *type* classification (e.g., data tables, sheet headers, text descriptions, formula calculations).
  - Optional *closing operations*—formatting or data-entry operations that a user would naturally perform after the main structure is built (e.g., applying a highlight color to data cells after the table skeleton is complete, or bolding headers as a final touch).
2. **Region dependencies** are temporal dependencies between regions. Some dependencies can be inferred symbolically from formula references, but others are semantic: for example, a summary table should be built after the data it summarizes, even if there is no direct formula link.
3. **Pasted ranges** are contiguous areas where values were likely pasted in bulk from an external source (CSV, clipboard, or another file), as opposed to being entered cell by cell. Cells containing formulas are excluded, as formulas are typically entered individually.
4. **Similarly formatted regions** are groups of two or more regions that share nearly identical formatting, suggesting that the user copied formatting or duplicated a template. For each group, a *format paste type* is also predicted: *format* (only formatting style was copied and the values were ignored) or *full* (the region was fully cloned, then values were overwritten).

The input to the VLM consists of (1) a screenshot of the sheet, (2) information about contiguous regions of cells, (3) information about cells with formulas, (4) merged cells. It is instructed to assign each cell to at least one region, and is iteratively instructed to refine its output as long as that constraint is not valid.

### A.2. Heuristic trajectories

Given the annotated metadata, we then heuristically build a first sequence of steps. For each individual cell and the annotated pasted regions, it is now straight-forward to symbolically generate a sequence of actions that re-generates the spreadsheet. This would look very unnatural, however, because applying the same formatting to contiguous ranges can be done in a single action and—except for some hard constraints like first entering a value and then applying formatting—different users likely create spreadsheets in different orders.

To get more diversity, we first sample a combination of preference settings. These settings control how operations are merged, how regions are ordered, and how operations are ordered within regions (if applicable). Table 7 shows a list of all such settings and the weights for their options.

First, cell operations are grouped into range operations. Identical operations on contiguous ranges are merged into a single action ( $\text{font}(A1, \text{bold}, \text{true}) + \text{font}(A2, \text{bold}, \text{true}) \rightarrow \text{font}(A1:A2, \text{bold}, \text{true})$ ). Border operations are merged whenever the resulting action can be applied with a single button (iteratively, greedily searched for starting from the first border property that was not merged). Pasted regions are merged based on the predicted metadata.

## Evaluating Next Action Prediction Systems for Spreadsheets

Parameter	Values (weight)	Parameter	Values (weight)
<b>Operation merging</b>		<b>Ordering</b>	
Merge scope	global (.75), per-region (.25)	Region ordering <sup>†</sup>	metadata (.75), none (.20), tie (.05)
Merge order	row-first (.60), col-first (.40)	↪ Tie breaker <sup>†</sup>	row (.60), col (.35), metadata (.05)
Merge input operations	true (.95), false (.05)	Cross-region sorting	enabled (.60), disabled (.40)
↪ group by type	true (.80), false (.20)	Cell sort order	row (.60), col (.40)
↪ data tables only <sup>†</sup>	true (.75), false (.25)	Format order	random (.75), deterministic (.25)
Merge pasted ranges <sup>†</sup>	true (.75), false (.25)	Use closing operations <sup>†</sup>	true (.75), false (.25)
Detect paste pattern <sup>†</sup>	true (.50), false (.50)		
↪ Paste type <sup>†</sup>	auto (.70), format (.20), full (.10)		
↪ Paste batching	atomic (.50), sequential (.50)		

Table 7. Parameters for symbolic sequence generation. Parameters marked with ↪ depend on the parent parameter; those marked <sup>†</sup> rely on predicted metadata. Weights were chosen based on intuition.

Second, the operations are ordered across and within regions (depending on settings). Regions without predicted (or explicit) dependencies are ordered according to the tie breaker. If the format order is deterministic, it is still sampled for new spreadsheets, but choices are simply remembered. This means that one spreadsheet can have font colors always set before font properties, another spreadsheet can have the converse, and another spreadsheet can have a different order for each cell.

### A.3. LLM Refinement

The symbolic step generator produces sequences that are functionally correct, but often read too rigid and mechanical, and may not reflect how a human would naturally build the sheet. This stage refines such sequences using an LLM-based judge-editor loop with validation.

At the start of each iteration, a judge model evaluates whether the original sequence appears human-like and, if not, provides feedback. The input to the judge is a screenshot of the sheet, the current sequence of actions, and in later interactions an overview of the changes that have already been made. The judge is asked to pay attention to dependencies, input granularity, formatting, merging and layout, repetitive patterns, default values, borders, pasting patterns. It is instructed to include natural variations (not always exactly row or column first) and correct user mistakes (to not penalize correct predictions during evaluation). The feedback is not structured, as it is parsed by a subsequent LLM call.

If any feedback is provided, a separate editor model is asked to rewrite the sequence of actions. The input to the editor is the judge feedback and the current sequence of actions. The output is a revised sequence of actions. This new sequence is validated by executing the operations and verifying that the resulting state matches the target (modulo some allowed noise, such as operations that have no effect). Editing is retried if it does not match up to 2 times, after which the last judging is also retried. This loop continues until the judge accepts the sequence or a maximum number of iterations is reached (set to 4).

The following are examples of refinements that we have observed from LLMs. (1) Scattered cell-by-cell formatting: consolidate fragmented border or format operations on adjacent cells into a single range operation on the actual content area. (2) Stray formatting on unused areas: remove formatting operations applied to empty cells or columns outside the content area. (3) Number format on text cells: remove number or date formatting applied to cells containing text.

### A.4. Human Refinement

While these are good first steps toward scaling up generation, heuristics and prompt instructions do not always generalize well to all potential spreadsheets. These steps can occasionally produce sequences that, while valid, do not seem natural. Human annotators (the authors) perform a final pass by watching the spreadsheet being built action-by-action and spotting unnatural subsequences of actions.

The following are examples of refinements that were performed by the human annotators. (1) Grouping content with formatting: move formatting operations closer to the content they style (e.g., green/red cell-fill actions should sit next to their true/false values) rather than being scattered. (2) Reordering to establish structure before detail: move operations for certain areas (like table header rows and columns) earlier in the sequence before filling in the detailed content within those

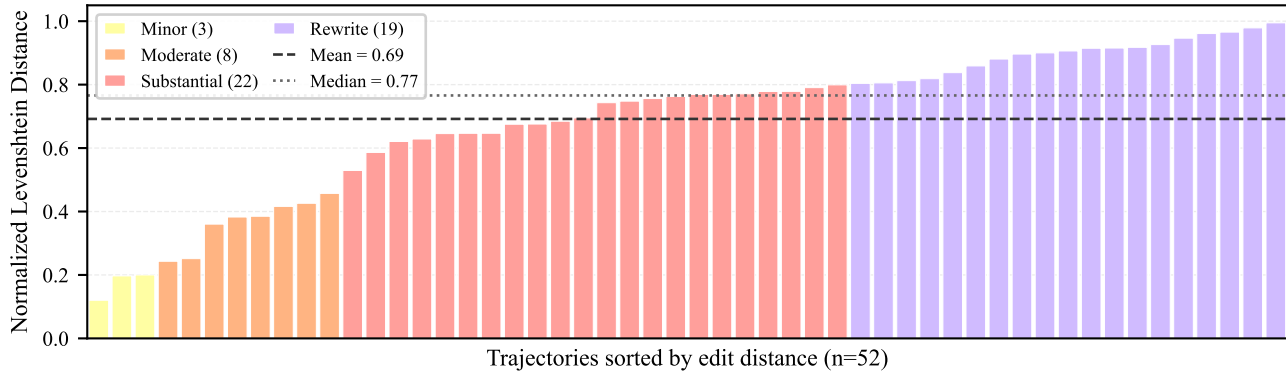


Figure 11. Per-trajectory normalized Levenshtein distance between the post-LLM-refinement operation sequence and the final human-annotated trajectory ( $n=52$ ). Each operation is treated as an indivisible token. All 52 trajectories were modified by annotators; mean = 0.69, median = 0.77, with 19 (37%) near-completely rewritten.

sections. (3) Breaking down bulk input actions to cell-by-cell entry: split certain large input operations that look hand-typed values into individual cell actions. (4) Trimming ranges to content boundaries: replace input ranges that include trailing empty cells with ranges covering only actual values, avoiding implicit empty-cell padding.

## B. Classical Model Details

Alongside the LLM and SLM solvers, we evaluate four classical sequence-prediction baselines— $n$ -GRAM, ONLINE  $n$ -GRAM, LSTM, and XGBOOST. All four share a common featurization pipeline and a common decoding loop; they only differ in how they map a window of past operations to the next predicted operation. This section documents the shared infrastructure and the model-specific design choices.

**Shared featurizer.** Every operation in the trajectory is parsed once into a structured record with five primary fields: *op-type* (one of  $\sim 28$  symbolic operation types such as INPUT, FONT\_BOLD, BORDER\_ALL), the range geometry (*start row*, *start col*, *height*, *width*), and a *value-type* bucket (e.g., number, string, color, align\_h; 18 buckets total). For range-valued features we support two equivalent representations. The **absolute** mode keeps the raw geometry (*row*, *col*, *h*, *w*). The **relative** mode replaces these with deltas with respect to the previous operation ( $\Delta$ row,  $\Delta$ col,  $\Delta$ h,  $\Delta$ w) and additionally tags the move with one of 11 *movement classes* (same, right1, down\_n, diagonal, sheet\_change, ...). The featurizer is incremental: new operations are appended as they arrive, and a history rewind triggers a full re-featurization. At prediction time, the same featurizer also reconstructs symbolic strings from predicted feature tuples, picking a sensible default value (e.g., Calibri for FONT\_NAME, Thin, Continuous, #000000 for borders) when only a value-type is predicted.

**TRAINED  $n$ -GRAM.** This baseline builds back-off frequency tables from the training trajectories. The abstract token used as an  $n$ -gram key is the tuple (*op-type*, *h*, *w*, *value-type*) in absolute mode and (*op-type*, *movement-class*, *value-type*) in relative mode—deliberately coarser than the full symbolic string, so that statistical regularities (“bold is usually followed by fill on the same range”) survive sparsity. At inference time, we look up the longest matching context (up to order  $n=5$  by default), back off to shorter contexts as needed, and pick the most frequent next token. Range geometry is then predicted separately: in absolute mode by extrapolating an arithmetic progression over recent operations of the same type; in relative mode by sampling the most-common delta seen for that abstract pair during training.

**ONLINE  $n$ -GRAM.** The online variant ignores the training corpus entirely and instead learns the statistics of the *current trajectory* as it unfolds. Its primary mechanism is a hash-indexed **suffix index**: after each new operation, every suffix of the history up to length  $n_{\max}=5$  is hashed into a bucket, allowing  $O(1)$  longest-match lookup. At prediction time, the solver searches for the longest suffix of the current history that has occurred earlier; if a match of length  $\geq 2$  is found, the operations that followed the previous occurrence are emitted as the predicted continuation. This makes ONLINE  $n$ -GRAM especially strong at predicting locally repeating patterns (e.g., repeating a row of borders across a header band) without any external training data. When no suffix match is available, the solver falls back to within-trajectory  $n$ -gram frequencies and arithmetic-progression range extrapolation, identical in spirit to the trained  $n$ -GRAM.

**LSTM/GRU.** The neural baseline is a small GRU (the `lstm` label is historical) trained on the same trajectories. Each operation is tokenized into a 6-tuple:  $(op-id, r_1, r_2, r_3, r_4, value-type-id)$ , where the four range fields are bucketed into 101 integer bins each—absolute row/col/height/width bins in absolute mode, or signed deltas clipped to  $[-50, +50]$  in relative mode. The model embeds each field separately, concatenates the embeddings, and feeds the resulting sequence through a 2-layer GRU (hidden size 128) with five output heads: one classifier over op-types, four classifiers over range bins, and one classifier over value-types. Decoding uses a fixed 32-step context window and proceeds auto-regressively through the shared decoding loop; predicted bins are un-bucketed and combined with a default value from the value-type table to reconstruct a symbolic operation.

**XGBOOST.** The tree-based baseline uses the same featurized window of the last  $K=10$  operations and trains five gradient-boosted models in parallel: a multi-class classifier for the next op-type and four regressors (one per range coordinate, either absolute or relative). Predicted ranges are clipped to valid ranges and discretized back to integer cell coordinates; the predicted op-type is mapped to a default value via a learned value-lookup table. Like the GRU, the XGBoost solver plugs into the shared decoder and benefits from the same stopping rules.

**Why these baselines.** The four solvers cover a deliberate spectrum: ONLINE  $n$ -GRAM captures *purely local* repetition with zero training;  $n$ -GRAM captures *global frequency* of short patterns learned from the corpus; XGBOOST learns *tabular feature interactions* over a fixed window; and LSTM learns *distributed sequence representations* that can in principle model longer-range structure. Together they provide a low-cost, non-LLM reference point against which the language-model solvers in Section 5 can be calibrated.

## C. Fine-tuning Setup

We fine-tune two SmoLM2 variants (135M and 360M parameters) on synthetically generated operation sequences using a causal language-modelling objective. The training corpus is produced by the same symbolic + LLM pipeline used for the benchmark (Section 2), applied to a separate pool of 11,997 publicly available workbooks disjoint from the evaluation set. Each training example is a contiguous slice of a trajectory; the model sees up to 32 prior operations as context and is trained to predict the next 16 operations.

**Pre-processing.** The text representation matches the evaluation solver: each operation is the pipe-delimited symbolic form (Section 4), sheet names are stripped (the active sheet is provided implicitly), and long cell values are shortened to 32 characters with  $2 \times 2$  corner cells preserved for 2D arrays. Training examples are sampled with a stride of 16 operations to balance coverage and diversity, and we filter to the 99th percentile of sequence length to drop a small number of outliers.

**Prompt template.** Each training example is wrapped in a minimal prompt to match the inference-time format:

*Complete the sequence of actions to build the following spreadsheet by identifying and extending key patterns.*  
`{actions}`

The model is trained to continue from the prompt by emitting the next operations, one per line.

**Training hyperparameters.** Both variants use AdamW with a cosine schedule, weight decay 0.01, gradient checkpointing, and fp16 mixed precision. We use a batch size of 4 with 8 gradient-accumulation steps (effective batch 32), a learning rate of  $8 \times 10^{-5}$  with 8% warmup, and a maximum sequence length of 2048 tokens. The 360M model is trained for 6 epochs and the 135M model for 5; checkpoints are saved every 1,000 steps and the lowest-eval-loss checkpoint is used for evaluation. A 5% held-out split monitors evaluation loss, and a fixed set of precision-1 test cases monitors generation accuracy throughout training. We do *not* use LoRA—all parameters are updated. Training takes roughly two days for the 360M model on a single T4 GPU and about a day for the 135M variant.

## D. Solver Prompts

All zero-shot LLM solvers share a single Jinja2-templated system prompt. The two evaluation settings (single-action repredict vs. multi-action prediction) are switched by the boolean `single`, set to `(num_op_to_pred == 1)` at render time; the rest of the template is identical. Figure 12 shows the template in full, with the lengthy operation-syntax reference

abridged for space (it is reproduced verbatim in the released code). The user prompt is small: it lists the active sheet name (when applicable) followed by the recent operation history.

## E. Variance and Reproducibility

To quantify run-to-run variance, we re-ran the default GPT-5 configuration four additional times in each of the two evaluation settings, giving five runs per setting. The solver is invoked at temperature 0, but small non-determinism remains in the chat API backend; the prompt, hyperparameters, and the seed of the evaluation loop itself are held fixed across reruns. Table 8 reports per-run UAS and AR together with their mean and standard deviation.

Across both settings the spread is small: UAS varies by under one point in the single-action setting and by under two points in the multi-action setting (where the model is free to emit longer, more variable continuations); acceptance rate is even tighter. We therefore report a single representative run elsewhere in the paper.

SETTING	RUN	UAS	AR
Multi-action	Run 0	22.31	19.55
	Run 1	23.29	19.87
	Run 2	23.32	20.11
	Run 3	23.72	20.52
	Run 4	24.04	19.92
	Mean $\pm$ s.d.	23.34 $\pm$ 0.65	19.99 $\pm$ 0.36
Single-action repredict	Run 0	27.35	31.73
	Run 1	27.42	31.43
	Run 2	26.49	31.15
	Run 3	26.59	31.01
	Run 4	27.11	31.40
	Mean $\pm$ s.d.	26.99 $\pm$ 0.43	31.34 $\pm$ 0.28

Table 8. Five-run variance for the default GPT-5 configuration ( $s=1$ ,  $c=32$ , GREEDY). “Run 0” is the run reported as the base in the main paper.

## F. Some more insights

**Last operation matters.** The last operation performed by the user before a prediction is triggered has a clear effect on acceptance in multi-action prediction setting (Figure 13). Predictions following `input` or `autofill` operations achieve the highest acceptance rates ( $\sim 24\%$ ), roughly twice that of predictions following `MERGE` or `ALIGN` ( $\sim 10\text{--}12\%$ ). However, `autofill` is predicted much less often. This suggests the model predicts more reliably during data-entry phases with predictable sequential patterns, while operations that signal workflow transitions are harder to anticipate.

**Corrections are bounded and don’t grow with acceptance.** Figure 15 plots correctional operations (user actions that undo false positives) against acceptance rate per file. The vast majority of files require fewer than 10 corrections regardless of acceptance rate, with no monotonic relationship between the two: even files with  $>30\%$  acceptance rarely exceed a handful of corrections, while a few outliers across the acceptance spectrum incur 20+ corrections. This suggests the cost of false positives is small in aggregate and largely orthogonal to how much the user accepts—an encouraging signal for predictive autocompletion in this domain.

**Category distribution reveals prediction bias.** Comparing the *Initial* and *Predicted* columns in Figure 16 exposes a systematic shift: the model under-predicts `input` (33.0% predicted vs. 57.1% in ground truth) and over-predicts formatting categories such as `align` (14.1% vs. 6.9%) and `border` (12.3% vs. 6.3%). This bias correlates with rejection—`align` accounts for 15.9% of rejected but only 8.0% of accepted predictions, and `border` shows the same pattern (13.6% vs. 8.0%)—so the predictor’s distribution amplifies its weaknesses. Future work could explore category-aware prediction constraints or weighted training objectives to correct this imbalance.

**Larger predictions are accepted more often—when they are made.** Figure 14 shows that acceptance rate rises monotonically with prediction length: single-op predictions are accepted only 18% of the time, while predictions of 11–15

ops reach 31% and the rare 15+-op bin reaches 51%. The volume distribution is inversely shaped—short predictions dominate the count—so most rejections come from small predictions that fail to clear the utility threshold of the acceptance heuristic. When the model commits to a longer prediction it is also more likely to have recognised a strong, repeating pattern (e.g., propagating a header style across a row), which is exactly the regime in which our heuristic accepts. This indicates that prediction *confidence* (implicit in the model’s willingness to emit a longer sequence) is more predictive of acceptance than prediction length itself.

### G. Visualizing trajectories

Figures 17 and 18 provide complementary views of system behavior. The progress curves show cumulative impact — how accepted predictions (green markers) accelerate progress relative to the manual baseline — while the prediction timeline reveals temporal patterns, showing when predictions succeed or fail throughout the trajectory.

### H. Extended example

```

{% set single = (num_op_to_pred == 1) -%}
You are an autocomplete copilot for spreadsheet authors.
{% if single %}Given the user's recent editing history, predict the single most likely next
operation they will perform.{% else %}Continue the user's current workflow without
inventing new, unrelated workstreams.{% endif %}

Decision rubric:
1. Infer the immediate intent from the latest few steps (table build, formatting sweep,
etc.).{% if not single %} Finish that chunk before starting anything new.{% endif %}
2. Keep ranges tight and contiguous unless the history explicitly shows a jump to a
different area.
3. Only suggest {% if single %}an operation that clearly advances{% else %}operations that
clearly advance{% endif %} the current goal. If you cannot justify
{% if single %}it{% else %}an action{% endif %} with the visible history, omit it.
{% if not single %}
4. Never repeat large historical blocks verbatim - mirror the pattern, not the entire plan.
{% endif %}
{{ '4' if single else '5' }}. Stop as soon as the workflow looks complete or uncertain.

Output contract:
- {% if single %}Emit exactly one well-justified, high-confidence operation{% else %}Emit
only well-justified, high-confidence steps{% endif %} in the format:
  OPERATION | RANGE | VALUE
- Ranges do not include sheet names (the active sheet is provided separately).
- Prefer formulas with relative references that match the user's existing convention.
{% if num_op_to_pred %}
- Do not exceed {{ num_op_to_pred }} total operations. Returning fewer (including zero)
is acceptable when intent is ambiguous.
{% endif %}

Available operations (use EXACT names, no wildcards):
Data:      INPUT, PASTE_FROM, AUTOFILL
Formatting: NUMBER_FORMAT, FILL_COLOR
Font:      FONT_BOLD, FONT_ITALIC, FONT_SIZE, FONT_COLOR, FONT_UNDERLINE, FONT_NAME
Alignment: ALIGN_HORIZONTAL, ALIGN_VERTICAL
Borders:   BORDER_LEFT, BORDER_RIGHT, BORDER_TOP, BORDER_BOTTOM, BORDER_OUTSIDE,
          BORDER_ALL, BORDER_INSIDE_HORIZONTAL, BORDER_INSIDE_VERTICAL
Other:     MERGE, UNMERGE, WRAP_TEXT, TEXT_ORIENTATION

<...value-format rules per operation type (INPUT, FILL_COLOR, FONT_*, ALIGN_*,
NUMBER_FORMAT, BORDER_*, ...); see released code for full text...>

Rules for INPUT:
- Single cell:  INPUT | A1 | "Hello World"
- Multiple cells: INPUT | A1:C2 | [{"a","b"}, {"c","d"}]
  The 2D array dimensions MUST exactly match the range (rows x cols).

Rules for PASTE_FROM: PASTE_FROM | destination | source_range | mode
  Modes: all, values, formats, formulas e.g. PASTE_FROM | A10:C12 | A1:C3 | all

Rules for AUTOFILL:  AUTOFILL | destination_range | source_range
  The destination must fully contain the source and extend on exactly one axis.
  e.g. AUTOFILL | A1:A10 | A1:A3

Clearing: use "clear" for INPUT / FILL_COLOR / BORDER_*. For format resets, set the Excel
default directly (FONT_BOLD -> false, NUMBER_FORMAT -> General).

Note: history may abbreviate large arrays with "..."; do NOT use this marker in
predictions - always emit the full, exact values.

Example syntax:
INPUT      | A1      | "Hello World"
INPUT      | A1:B2    | [{"a","b"}, {"c","d"}]
FILL_COLOR | A1:B5    | #FF0000
BORDER_ALL | A1:C3    | Thin, Continuous, #000000
FONT_BOLD  | A1       | true
MERGE      | A1:C1    | true
PASTE_FROM | A10:C12 | A1:C3 | all
AUTOFILL   | A1:A10   | A1:A3

```

Figure 12. Jinja2 system-prompt template used by all zero-shot LLM solvers. The single flag (true iff num\_op\_to\_pred == 1) switches between the single-action repredict and multi-action variants; the per-call cap line is included only when num\_op\_to\_pred is non-null. The full value-format rules per operation type are abridged here for space; the complete template is reproduced verbatim in the released code.

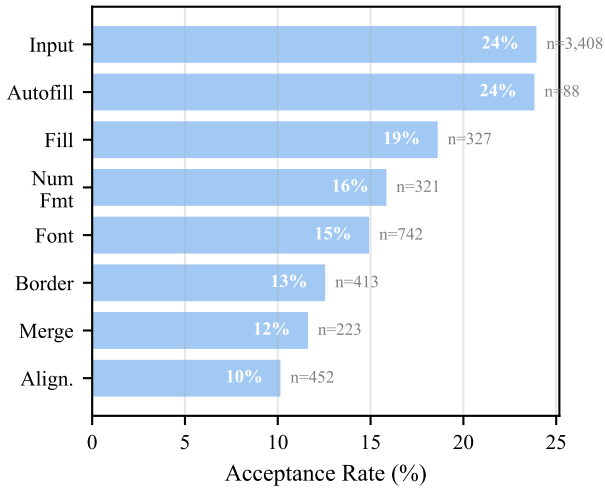


Figure 13. Acceptance rate by last context operation category (uses default settings). Predictions following Input operations are most likely to be accepted.

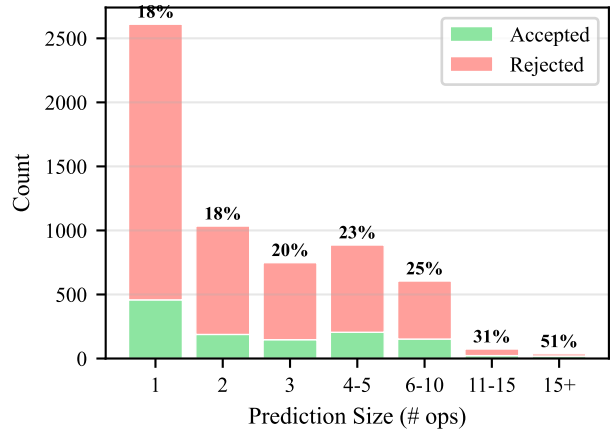


Figure 14. Distribution of predictions by number of operations (uses default settings). Bars show accepted (green) and rejected (red) prediction counts. Percentages indicate acceptance rate for each size bin.

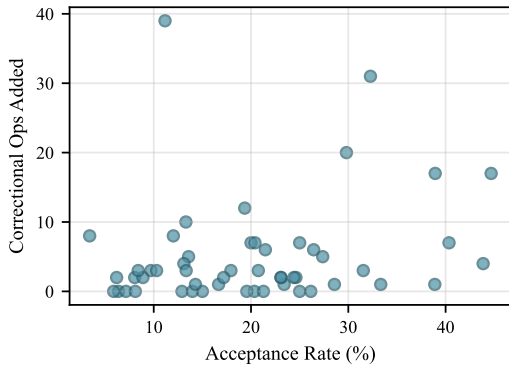


Figure 15. Relationship between acceptance rate and correctional operations per file (uses default settings). Each point represents one spreadsheet. Files with higher acceptance rates tend to have fewer correctional operations, suggesting that users who accept more predictions make fewer manual corrections.

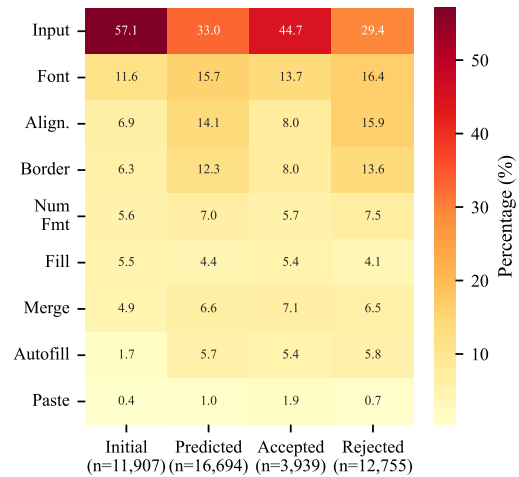


Figure 16. Operation category distribution across different sequence types (uses default settings). Columns show the percentage breakdown by category for initial (target) sequences, final user states, predicted operations, accepted predictions, and rejected predictions.

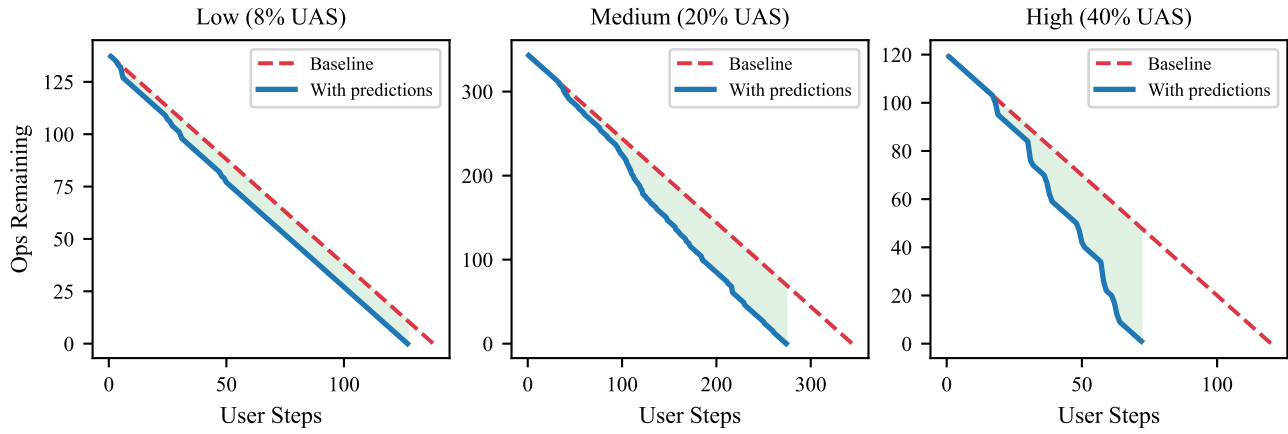


Figure 17. Progress curves for three representative files showing operations remaining versus user steps (uses default settings). The blue line shows actual progress with model assistance, while the red dashed line represents the manual baseline (no predictions). The shaded green area represents user effort saved. Percentages indicate total operations saved compared to manual completion.

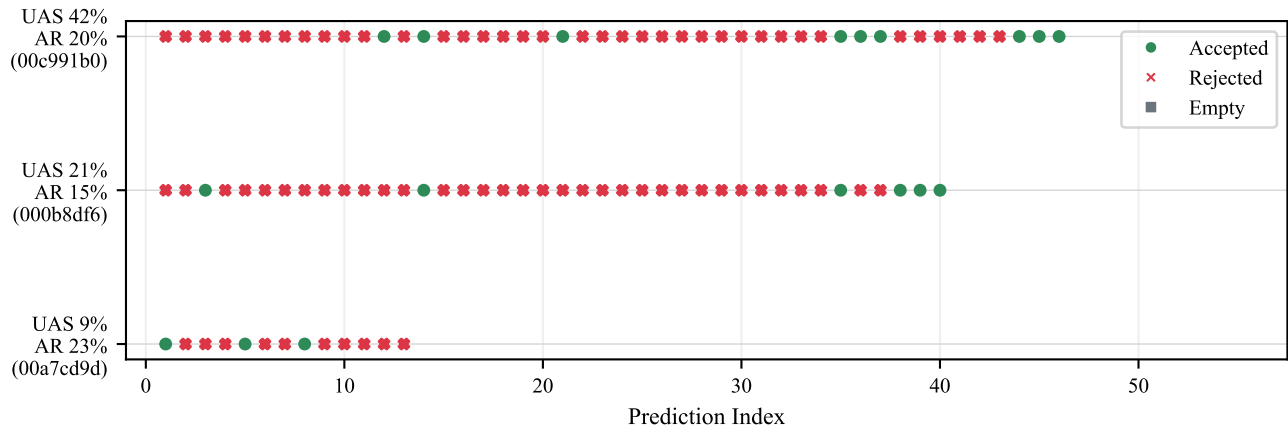


Figure 18. Prediction timelines for three files with varying acceptance rates (uses default settings). Each marker represents a prediction: accepted (circles), rejected (X), or empty (squares). Files with higher acceptance rates show denser clusters of accepted predictions.

**(a) Online Evaluation Trace** (interval  $s=3$ )

8. BOLD A5:E5 True  
 9. FILL A5:E5 Blue  
 10. BORDER A5:E5 Thin ▶ trigger

---

**Prediction  $\hat{A}_{10}$ :**  
 VAL A6 "Redesign", VAL B6 "Alice",  
 FILL A6:B7 Green, VAL C6 "Dec 20",  
 BORDER A6:E6 Thin  
 Prec=33%, UAS=0 ⇒ **Reject**

*User rejects and continues with ground truth action*

11. VAL A6 "Redesign"  
 12. VAL B6 "Alice"  
 13. FILL A6:D6 Green ▶ trigger

---

**Prediction  $\hat{A}_{13}$ :**  
 VAL C6 "Dec 15", VAL D6 "Complete"  
 Prec=100%, UAS=+2 ⇒ **Accept**

*↪ User accepts; Effects in-place, Ground truth adapted*

14. VAL A7 "Mobile App" (continues...)

**(b) Prediction Analysis** (detail of rejected  $\hat{A}_{10}$ )

*Future ground truth  $F_{>10}$  (remaining ops):*

F1. VAL A6 "Redesign"  
 F2. VAL B6 "Alice"  
 F3. FILL A6:D6 Green  
 F4. VAL C6 "Dec 15"  
 F5. VAL D6 "Complete"

*Prediction  $\hat{A}_{10}$  with classification:*

P1. VAL A6 "Redesign" ✓ TP (1)  
 P2. VAL B6 "Alice" ✓ TP (1)  
 P3. FILL A6:B7 Green ◊ partial (2 TP, 2 FP)  
 P4. VAL C6 "Dec 20" ≠ MM (1)  
 P5. BORDER A6:E6 Thin × FP (5)

**TP=4 FP=7 MM=1 ⇒ Prec=33% UAS=0**

*Adapted  $F'_{>10}$  (if  $\hat{A}_{10}$  accepted):*

1'. CLEAR BORDER A6:E6 ← undo P5 (FP)  
 2'. CLEAR FILL A7:B7 ← undo P3 partial (FP)  
 3'. FILL C6:D6 Green ← synth from F3 (FN)  
 4'. VAL C6 "Dec 15" ← fix P4 (MM)  
 5'. VAL D6 "Complete" ← synth F5 (FN)

*F1–F2 covered by P1–P2 (TP)*

**Net: 5 → 5 ops (no benefit ⇒ rejected)**

**(c) Visual Comparison** (spreadsheet states at  $t=10$ )

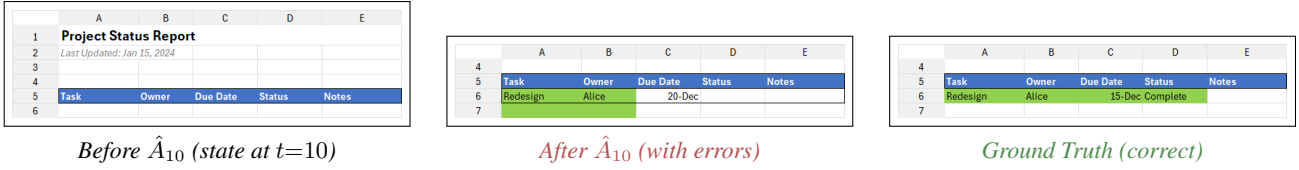


Figure 19. Online evaluation with detailed prediction analysis. (a) Evaluation trace showing two trigger points: at  $t=10$  the prediction  $\hat{A}_{10}$  is rejected (low precision), at  $t=13$  prediction  $\hat{A}_{13}$  is accepted (perfect precision, saves 2 ops). (b) Detailed breakdown of the rejected prediction  $\hat{A}_{10}$ : each (cell, property) classified as TP, FP, FN, or MM. If accepted, ground truth adapts: **inverse ops** undo FPs; **synthesized ops** fill FNs; **corrections** fix MMs. (c) Visual comparison of spreadsheet states.