Soylent: A Word Processor with a Crowd Inside

Michael S. Bernstein[1], Greg Little[1], Robert C. Miller[1],
Björn Hartmann[2], Mark S. Ackerman[3], David R. Karger[1], David Crowell[1], Katrina Panovich[1]

[1] MIT CSAIL	[2] Computer Science Division	[3] Computer Science & Engineering
Cambridge, MA	University of California, Berkeley	University of Michigan
{msbernst, glittle, rcm,	Berkeley, CA	Ann Arbor, MI
karger, dcrowell, kp}@csail.mit.edu	bjoern@cs.berkeley.edu	ackerm@umich.edu

ABSTRACT

This paper introduces architectural and interaction patterns for integrating crowdsourced human contributions directly into user interfaces. We focus on writing and editing, complex endeavors that span many levels of conceptual and pragmatic activity. Authoring tools offer help with pragmatics, but for higher-level help, writers commonly turn to other people. We thus present Soylent, a word processing interface that enables writers to call on Mechanical Turk workers to shorten, proofread, and otherwise edit parts of their documents on demand. To improve worker quality, we introduce the Find-Fix-Verify crowd programming pattern, which splits tasks into a series of generation and review stages. Evaluation studies demonstrate the feasibility of crowdsourced editing and investigate questions of reliability, cost, wait time, and work time for edits.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Design**,** Human Factors

Keywords: Outsourcing, Mechanical Turk, Crowdsourcing

INTRODUCTION

Word processing is a complex task that touches on many goals of human-computer interaction. It supports a deep cognitive activity – writing – and requires complicated manipulations. Writing is difficult: even experts routinely make style, grammar and spelling mistakes. Then, when a writer makes high-level decisions like changing a passage from past to present tense or fleshing out citation sketches into a true references section, she is faced with executing daunting numbers of nontrivial tasks across the entire document. Finally, when the document is a half-page over length, interactive software provides little support to help us trim those last few paragraphs. Good user interfaces aid these tasks; good artificial intelligence helps as well, but it is clear that we have far to go.

In our everyday life, when we need help with complex cognition and manipulation tasks, we often turn to other people. We ask friends to answer questions that we cannot answer ourselves [8]; masses of volunteer editors flag spam

edits on Wikipedia [13]. Writing is no exception [7]: we commonly recruit friends and colleagues to help us shape and polish our writing. But we cannot always rely on them: colleagues do not want to proofread every sentence we write, cut a few lines from every paragraph in a ten-page paper, or help us format thirty ACM-style references.

As a step toward integrating this human expertise permanently into our writing tools, we present *Soylent*, a word processing interface that utilizes crowd contributions to aid complex writing tasks ranging from error prevention and paragraph shortening to automation of tasks like citation searches and tense changes. We hypothesize that crowd workers with a basic knowledge of written English can support both novice and expert writers. These workers perform tasks that the writer might not, such as scrupulously scanning for text to cut, or updating a list of addresses to include a zip code. They can also solve problems that artificial intelligence cannot, for example flagging writing errors that the word processor does not catch.

Soylent aids the writing process by integrating paid crowd workers from Amazon's Mechanical Turk platform[1] into Microsoft Word. *Soylent is people*: its core algorithms involve calls to Mechanical Turk workers (Turkers). Soylent is comprised of three main components:

1) *Shortn*, a text shortening service that cuts selected text down to 85% of its original length typically without changing the meaning of the text or introducing errors.
2) *Crowdproof*, a human-powered spelling and grammar checker that finds problems Word misses, explains the problems, and suggests fixes.
3) *The Human Macro*, an interface for offloading arbitrary word processing tasks such as formatting citations or finding appropriate figures.

The main contribution of this paper is *the idea of embedding paid crowd workers in an interactive user interface to support complex cognition and manipulation tasks on demand.* These crowd workers do tasks that computers cannot reliably do automatically and the user cannot easily script. This paper contributes the design of one such system, an implementation embedded in Microsoft Word, and a programming pattern that increases the reliability of paid crowd workers on complex tasks. We expand these contributions with feasibility studies of the performance, cost, and time delay of our three main components and a discus-

---

[1] http://www.mturk.com

sion of the limitations of our approach with respect to privacy, delay, cost, and domain knowledge.

The fundamental technical contribution of this work is a crowd programming pattern called *Find-Fix-Verify*. Mechanical Turk costs money and it can be error-prone; to be worthwhile to the user, we must control costs and ensure correctness. Find-Fix-Verify *splits complex crowd intelligence tasks into a series of generation and review stages that utilize independent agreement and voting to produce reliable results*. Rather than ask a single crowd worker to read and edit an entire paragraph, for example, Find-Fix-Verify recruits one set of workers to find candidate areas for improvement, then collects a set of candidate improvements, and finally filters out incorrect candidates. This process prevents errant crowd workers from contributing too much, too little, or introducing errors into the document.

Soylent is influenced by prior work on crowdsourced interfaces (e.g., [1, 9, 24]). Such work has generally aggregated previous crowd interactions rather than recruited an on-demand workforce for new requests. Instead of training on previous users, we ask crowd workers to solve personalized tasks on demand each time. This interface-for-hire model has benefits and limitations that we explore in this paper.

In the rest of this paper, we review related work in crowdsourced interfaces and text editing. We then introduce Soylent and its main components: Shortn, Crowdproof, and The Human Macro. We detail the Find-Fix-Verify pattern that powers Soylent; evaluate the feasibility of our three components; and conclude with a discussion of privacy issues and inherent limitations of our approach.

RELATED WORK

Soylent is related to work in two areas: crowdsourcing systems, and artificial intelligence for word processing.

Crowdsourcing

Gathering data to train algorithms is a common use of crowdsourcing. For example, the ESP Game [27] collects descriptions of objects in images for use in object recognition. Mechanical Turk is already used to collect labeled data for machine vision [26] and natural language processing [25]. Soylent tackles problems that are currently infeasible for AI algorithms, even with abundant data. However, Soylent's output may be used to train future AIs.

Several systems power novel interactions with the wisdom of crowds. HelpMeOut [9] collects debugging traces and applies others' error solutions to help fix code. FeedMe [1] and Collabio [2] use friends to power recommender systems and tag cloud visualizations. MySong [24] indexes a library of music chords to enable the user to build a chord progression by singing a melody line. Google Suggest mines the search engine's query logs to speed and direct new queries. Soylent is unique in asking paid crowd workers to solve the user's problems, rather than aggregating past activity. Having an on-demand workforce also expands the realm of tasks we can support beyond those requiring user traces or incentives.

Soylent builds on work embedding on-demand workforces inside applications and services. ChaCha[2] recruits humans to do search engine queries for users who are mobile; Amazon Remembers uses Mechanical Turk to find products that match a photo taken by the user on a phone; Sala et al.'s PEST [23] uses Mechanical Turk to vet advertisement recommendations. These systems consist of a single user operation and little or no interaction. Soylent extends this work to more creative, complex tasks where the user can make personalized requests and interact with the returned data by direct manipulation.

Proofreading is emerging as a common task on Mechanical Turk. Standard Minds[3] offers a proofreading service backed by Mechanical Turk that accepts plain text via a web form and returns edits one day later. By contrast, Soylent is embedded in a word processor, has much lower latency, and presents the edits in Microsoft Word's user interface. Our work also contributes the Find-Fix-Verify pattern to improve the quality of such proofreading services.

Soylent's usage of human computation means that its behavior depends in large part on qualities of crowdsourcing systems and Mechanical Turk in particular. Recently, Ross et al. found that Mechanical Turk had two major populations: well-educated, moderate-income Americans, and young, well-educated but less wealthy workers from India [22]. Kittur and Chi considered how to run user studies on Mechanical Turk, proposing the use of quantitative verifiable questions as a verification mechanism [11]. Find-Fix-Verify builds on this notion of requiring verification to control quality. Heer and Bostock explored Mechanical Turk as a testbed for graphical perception experiments, finding reliable results when they implemented basic measures like qualification tests [10]. Little et al. advocate the use of human computation algorithms on Mechanical Turk [16]. Find-Fix-Verify may be viewed as a new design pattern for human computation algorithms. It is specifically intended to control lazy and overeager Turkers, identify which edits are tied to the same problem, and visualize them in an interface. Quinn and Bederson have authored a survey of human computation systems that expands on this brief review [21].

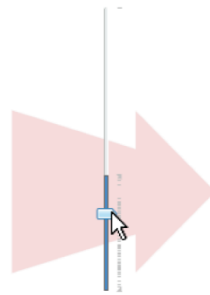### Artificial Intelligence for Word Processing

Automatic proofreading has a long history of research [14] and has seen successful deployment in word processors. However, Microsoft Word's spell checker frequently suffers from false positives, particularly with proper nouns and unusual names. Its grammar checker suffers from the opposite problem: it misses blatant errors.[4] Human checkers are currently more reliable, and can also offer suggestions on how to fix the errors they find, which is not always possible for Word — for example, the common (but useless) Microsoft Word feedback, "Fragment; consider revising."

---

[2] http://www.chacha.com
[3] http://www.standardminds.com
[4] http://faculty.washington.edu/sandeep/check

Figure 1. Shortn allows users to adjust the length of a paragraph via a slider. Red text indicates locations where cuts or rewrites have occurred. Tick marks represent possible lengths, and the blue background bounds the possible lengths.

Soylent's Shortn component is related to document summarization, which has also received substantial research attention [18]. Microsoft Word has a summarization feature that uses *sentence extraction,* which identifies whole sentences to preserve in a passage and deletes the rest, producing substantial shortening but at a great cost in content. Shortn's approach, which can rewrite or cut parts of sentences, is an example of *sentence compression*, an area of active recent research [5, 12] that suffers from a lack of training data [4].

The Human Macro is related to AI techniques for end-user programming. Several systems allow users to demonstrate repetitive editing tasks for automatic execution; examples include Eager, TELS, and Cima [6], LAPIS [20], and SmartEdit [15]. Other work has considered natural-language-like programming syntax (e.g. [17]).

SOYLENT
Soylent is a prototype crowdsourced word processing interface with three features: shortening, proofreading, and arbitrary macro tasks via human-language input.

Shortn: Text Shortening
Some authors struggle to remain within length limits on papers and spend the last hours of the writing process tweaking paragraphs to shave a few lines. This is painful work and a questionable use of the authors' time. Other writers write overly wordy prose and need help editing. Automatic summarization algorithms can provide useful summaries [18], but cannot easily determine what language to cut or shorten. Additionally, they cannot use language generation techniques to make sure the resulting text flows.

Soylent's Shortn interface allows authors to condense sections of text. The user selects the area of text that is too long—for example a paragraph or section—then presses the Shortn button in the Word's Soylent ribbon tab. In response, Soylent launches a series of Mechanical Turk tasks in the background and notifies the user when the text is ready. The user can then launch the Shortn dialog box

(Figure 1). On the left is the original paragraph; on the right is the proposed revision. Shortn provides a single slider to allow the user to continuously adjust the length of the paragraph. As the user does so, Shortn computes the combination of crowd trimmings that most closely match the desired length and presents that text to the user on the right. From the user's point of view, as she moves the slider to make the paragraph shorter, sentences are slightly edited, combined and cut completely to match the length requirement. Areas of text that have been edited or removed are highlighted in red in the visualization. These areas may differ from one slider position to the next: the cuts are not monotonic in this sense.

Shortn typically can remove up to 15–30% of a paragraph in a single pass, and up to ~50% with multiple iterations. The Shortn algorithm preserves meaning when possible, so it cuts unnecessary language or concept repetition. Removing whole arguments or sections is left to the user.

Crowdproof: Crowdsourced Proofreading
Soylent provides a human-aided spelling, grammar and style checking interface called Crowdproof (Figure 2). Crowdproof aims to catch spelling, style and grammar errors that AI algorithms today cannot find or fix. The process finds errors, explains the problem, and offers one to five alternative rewrites. Crowdproof is essentially a distributed proofreader operating for cents per task.

To use Crowdproof, the user highlights a section of text and presses the proofreading button in the Soylent ribbon tab. The task is queued to the Soylent status pane and the user is free to keep working. (Because Crowdproof costs money, it does not issue requests unless commanded.)

When the crowd is finished, Soylent calls out the edited sections with a purple dashed underline. If the user clicks on the error, a drop-down menu explains the problem and offers a list of alternatives. By clicking on the desired alternative, the user replaces the incorrect text with an option of
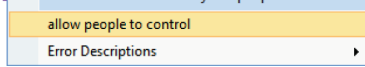
Figure 2. Crowdproof is a human-augmented proofreader. The drop-down explains the problem (blue title) and suggests fixes (gold selection).

Figure 3. The Human Macro is an end-user programming interface for automating document manipulations. The left half is the user's authoring interface; the right half is a preview of what the Turker will see.

his or her choice. If the user hovers over the Error Descriptions menu item, the popout menu suggests additional second-opinions of why the error was called out.

The Human Macro: Natural Language Crowd Scripting

Embedding crowd workers in an interface allows us to reconsider designs for short end-user programming tasks. Typically, users need to translate their intentions into algorithmic thinking explicitly via a scripting language or implicitly through learned activity [6]. But tasks conveyed to humans can be written in a much more natural way. While natural language command interfaces continue to struggle with unconstrained input over a large search space, humans are good at understanding written instructions.

The Human Macro is Soylent's natural language command interface. Soylent users can use it to request arbitrary work quickly in human language. Launching the Human Macro opens a request form (Figure 3). The design challenge here is to ensure that the user creates tasks that are scoped correctly for a Mechanical Turk worker. We wish to prevent the user from spending money on a buggy command.

The form dialog is split in two mirrored pieces: a task entry form on the left, and a preview of what the Turker will see on the right. The preview contextualizes the user's request, reminding the user he is writing something akin to a Help Wanted or Craigslist advertisement. The form suggests that the user provide an example input and output, which is an effective way to clarify the task requirements to workers. If the user selected text before opening the dialog, he has the option to split the task by each sentence or paragraph, so (for example) the task might be parallelized across all entries on a list. The user then chooses how many separate Turkers he would like to complete the task. The Human Macro helps debug the task by allowing a test run on one sentence or paragraph.

The user chooses whether the Turkers' work should replace the existing text or just annotate it. If the user chooses to replace, the Human Macro underlines the text in purple and enables drop-down substitution like the Crowdproof interface. If the user chooses to annotate, the feedback populates comment bubbles anchored on the selected text by utilizing Word's reviewing comments interface.

TECHNIQUES FOR PROGRAMMING CROWDS

This section characterizes the challenges of leveraging crowd labor for open-ended document editing tasks. We introduce the Find-Fix-Verify pattern to improve output quality in the face of uncertain worker quality. Over the past year, we have performed and documented dozens of experiments on Mechanical Turk.[5] For this project alone, we have interacted with 8809 Turkers across 2256 different tasks. We draw on this experience in the sections to follow.
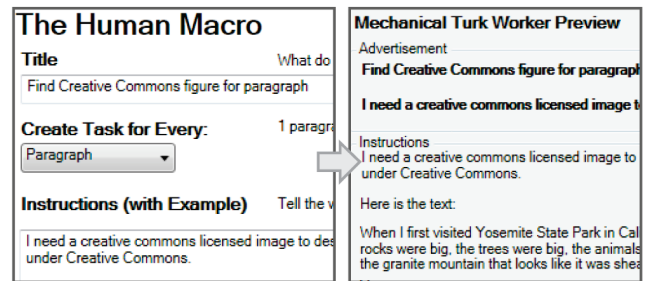
Challenges in Programming with Crowd Workers

We are primarily concerned with tasks where workers directly edit a user's data in an open-ended manner. These tasks include shortening, proofreading, and user-requested changes such as address formatting. In our experiments, it is evident that many of the raw results that Turkers produce on such tasks are unsatisfactory. As a rule-of-thumb, roughly 30% of the results from open-ended tasks are poor. This "30% rule" is supported by the experimental section of this paper as well. Clearly, a 30% error rate is unacceptable to the end user. To address the problem, it is important to understand the nature of unsatisfactory responses.

High Variance of Effort

Turkers exhibit high variance in the amount of effort they invest in a task. We might characterize two useful personas at the ends of the effort spectrum, the *Lazy Turker* and the *Eager Beaver*. The *Lazy Turker* does as little work as necessary to get paid. For example, when asked to proofread the following error-filled paragraph from a high school essay site,[6] a Lazy Turker inserted only a single character to correct a spelling mistake. The change is highlighted:

```
The theme of loneliness features throughout many scenes in Of Mice and
Men and is often the dominant theme of sections during this story. This
theme occurs during many circumstances but is not present from start to
finish. In my mind for a theme to be pervasive is must be present during
every element of the story. There are many themes that are present most
of the way through such as sacrifice, friendship and comradeship. But in
my opinion there is only one theme that is present from beginning to
end, this theme is pursuit of dreams.
```

A first challenge is thus to discourage or prevent workers from such behavior. Kittur et al. attacked the problem of Lazy Turkers in crowdsourced user studies [12] by adding clearly verifiable, quantitative questions (e.g., "How many sections does the article have?") that forced the Lazy Turker to read the material being studied.

Equally problematic as Lazy Turkers are *Eager Beavers*. Eager Beavers go beyond the task requirements in order to

be helpful, but create further work for the user in the process. For example, when asked to reword a phrase, one Eager Beaver provided a litany of options:

```
The theme of loneliness features throughout many scenes in Of Mice and
Men and is often the principal, significant, primary, preeminent, pre-
vailing, foremost, essential, crucial, vital, critical theme of sections
during this story.
```

In their zeal, this worker rendered the resulting sentence ungrammatical. Eager Beavers may also leave extra comments in the document or reformat paragraphs. It would be problematic to funnel such work back to the user.

Both the Lazy Turker and the Eager Beaver are looking for a way to clearly signal to the requester that they have completed the work. Without clear guidelines, the Lazy Turker will choose the path that produces *any* signal and the Eager Beaver will produce too many signals.

Turkers Introduce Errors

Turkers working on complex tasks can accidentally introduce substantial new errors. For example, when proofreading paragraphs about the novel *Of Mice and Men*, Turkers variously changed the title to just *Of Mice*, replaced existing grammar errors with new errors of their own, and changed the text to state that *Of Mice and Men* is a movie rather than a novel. Such errors are compounded if the output of one Turker is used as input for other Turkers.

The Find-Fix-Verify Pattern

Our crowdsourced interface algorithms must control the efforts of both the Eager Beaver and Lazy Turker and limit introduction of errors. Absent suitable control techniques, the rate of problematic edits is too high to be useful. We feel that the state of programming crowds is analogous to that of UI technology before the introduction of design patterns like Model-View-Controller, which codified best practices. In this section, we propose the Find-Fix-Verify pattern as one method of programming crowds to reliably complete open-ended tasks that directly edit the user's data. We describe the pattern and then explain its use in Soylent.

Find-Fix-Verify

The Find-Fix-Verify pattern separates open-ended tasks into three stages where each worker can make a clear contribution. The first stage, Find, asks Turkers to identify *patches* of the user's work that need more attention. For example, when proofreading, the Find stage asks Turkers to highlight at least one phrase or sentence that needs editing (Figure 4). Any single Turker may produce a noisy result – for example, Lazy Turkers may prefer errors near the beginning of a paragraph. The Find stage aggregates independent opinions to find the most consistently cited problems: multiple independent agreement is typically a strong signal that a crowd is correct. Soylent keeps the patches (text ranges) that at least 20% of the workers agree on. Identified patches are then fed in parallel into the Fix stage.

The Fix stage recruits workers to revise an identified patch. Each task now consists of a constrained edit to an area of interest. The worker can see the entire paragraph but only edit the sentence or sentences directly containing the patch. A small number (3–5) of Mechanical Turk workers propose revisions. Even if 30% of work is bad, 3–5 submissions are sufficient to produce a few viable alternatives.

The Verify stage performs quality control on revisions. We randomize the order of the unique alternatives generated in the Fix stage and ask 3–5 new workers to vote on them (Figure 4). We either ask Turkers to vote on the best option (when the interface needs a default choice, like Crowdproof) or to flag poor suggestions (when the interface requires as many options as possible, like Shortn). To ensure that Turkers cannot vote for their own work, we ban all Fix workers from participating in the Verify stage.

Pattern Discussion

Why should tasks be split into independent Find-Fix-Verify stages? Why not let Turkers find an error and fix it, for increased efficiency and economy? Lazy Turkers will always choose the easiest error to fix, so combining Find and Fix will result in poor coverage. By splitting Find from Fix, we can direct Lazy Turkers to propose a fix to patches that they might otherwise ignore. Additionally, splitting Find and Fix enables us to merge work completed in parallel. Had each Turker edited the entire paragraph, we would not know which edits were trying to fix the same problem. By splitting Find and Fix, we can map edits to patches and produce a much richer user interface—for example, the multiple options in Crowdproof's replacement dropdown.

The Verify stage reduces noise in the returned result. Anecdotally, Turkers are better at vetting suggestions than they are at producing original work. Independent agreement among Verify workers can help certify an edit as good or bad. Verification trades off time lag with quality: a user who can tolerate more error but needs less time lag might opt not to verify work or use fewer verification workers.

One challenge that the Find-Fix-Verify pattern shares with other Mechanical Turk algorithms is that it can stall when workers are slow to accept the task. Rather than wait for ten Turkers to complete the Find task before moving on to Fix, a timeout parameter can force our algorithm to advance if a minimum threshold of workers have completed the work.

Find-Fix-Verify in Soylent

Both Shortn and Crowdproof use the Find-Fix-Verify pattern. We will use Shortn as an illustrative example. To provide the user with near-continuous control of paragraph length, Shortn should produce many alternative rewrites without changing the meaning of the original text or introduce[7] grammatical errors.

We begin by splitting the input region into paragraphs. The Find stage asks ten Turkers to identify candidate areas for shortening in each paragraph. At least 20% (i.e., two) of the Turkers must agree on a text region. Each agreed-upon patch moves on to the Fix stage, where five Turkers see the patch highlighted in the paragraph and are asked to shorten the patch. Each worker also votes whether the patch could

---

[7] Word's grammar checker, eight authors and six reviewers did not catch the error in this sentence. Crowdproof later did, and correctly suggested that "introduce" should be "introducing".

Microsoft Word
C# and Visual Studio Tools for Office

Mechanical Turk
Javascript, Java and TurKit

Soylent is a prototype crowdsourced word processing interface. It focuses on three main tasks: shortening the user's writing, proofreading [...]

`shorten(text)`

User selects text

Displayed to the user

Soylent, a prototype crowdsourced word processing interface, focuses on three tasks: shortening the user's writing, proofreading [...]

`return(patches)`

Find
"Identify at least one area that can be shortened without changing the meaning of the paragraph.

Find overlapping areas (patches)

Fix
"Edit the highlighted section to shorten its length without changing the meaning of the paragraph:"

Soylent, a prototype...

Randomize order of suggestions

Verify
"Choose at least one rewrite that has significant style errors in it. Choose at least one rewrite that significantly changes the meaning of the sentence."
☐ Soylent ~~is~~, a prototype...
☐ Soylent ~~is a~~ prototypes...
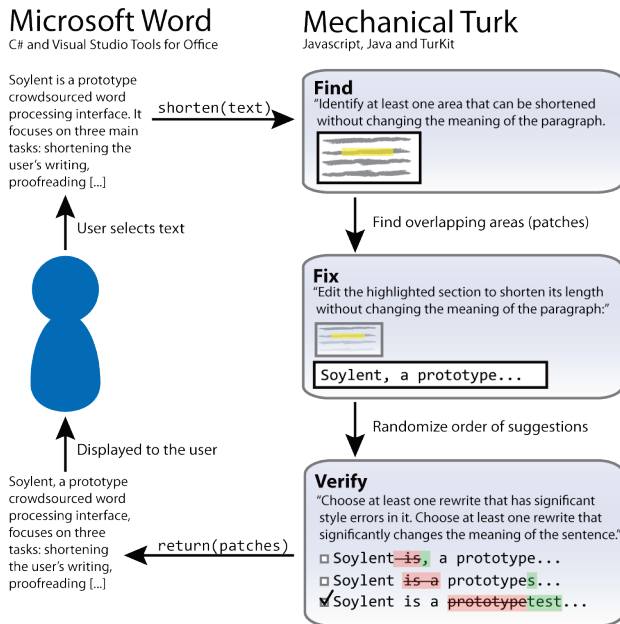☑ Soylent is a ~~prototype~~test...

Figure 4. Find-Fix-Verify identifies patches in need of editing, recruits workers to fix the patches, and votes to approve work.

be cut entirely. The algorithm now has a set of rewrites and votes on whether the text can be cut. If the patch can be cut, we introduce the empty string as a rewrite. In the Verify stage, five Turkers see a list of all the rewrites where each rewrite has been annotated using color and strikethroughs to highlight its differences from the original. Each Turker selects at least one rewrite that has significant spelling, grammar, or style problems, and at least one rewrite that significantly changes the meaning of the original sentence. We use majority voting to remove problematic rewrites and to decide whether the patch can be cut entirely. At the conclusion of the Verify stage, we have a set of candidate patches and a list of verified rewrites for each patch.

To keep the algorithm responsive, we use a 15-minute timeout at each stage. We require a minimum of six workers in Find, three workers in Fix, and three workers in Verify. When the user specifies a desired maximum length, Shortn searches for the longest combination of rewrites subject to the length constraint. This search is a special case of the knapsack problem and can be solved with a polynomial time dynamic programming algorithm.

IMPLEMENTATION
Soylent consists of a front-end application-level add-in to Microsoft Word and a back-end service to run Mechanical Turk tasks (Figure 4). The Microsoft Word plug-in is written using Microsoft Visual Studio Tools for Office (VSTO) and the Windows Presentation Foundation (WPF). Back-end scripts use the TurKit Mechanical Turk toolkit [16].

EVALUATION
Our initial evaluation sought to establish evidence for Soylent's end-to-end feasibility, as well as to understand the properties of the Find-Fix-Verify design pattern.

Shortn Evaluation
We evaluated Shortn quantitatively by running it on example texts. Our goal was to see how much Shortn could shorten text, as well as its associated cost and time characteristics. We collected five examples of texts that might be sent to Shortn, each between one and seven paragraphs long. We chose these inputs to span from preliminary drafts to finished essays and from easily understood to dense technical material (Table I).

To simulate a real-world deployment, we ran the algorithms with a timeout enabled and set to twenty minutes. We required 6–10 workers to complete the Find tasks and 3–5 workers to complete the Fix and Verify tasks: if a Find task failed to recruit even six workers, it might wait indefinitely. To be slightly generous while matching going rates on Mechanical Turk, we paid $0.08 per Find, $0.05 per Fix, and $0.04 per Verify.

Each resulting paragraph had many possible variations depending on the number of shortened alternatives that passed the Verify stage – we chose the shortest possible version for analysis and compared its length to the original paragraph. We also measured *wait time*, the time between posting the task and the worker accepting the task, and *work time*, the time between acceptance and submission. In all tasks, it was possible for the algorithm to stall while waiting for workers, having a large effect on averages. Therefore, we report medians, which are more robust to outliers.

Results
Shortn produced revisions that were 78%–90% of the original document length. For reference, a reduction to 85% could slim an 11¾ page UIST draft down to 10 pages with no substantial cuts in the content. Table I summarizes and gives examples of Shortn's behavior. Typically, Shortn focused on unnecessarily wordy phrases like "are going to have to" (Table I, Blog). Turkers merged sentences when patches spanned sentence boundaries (Table I, Classic UIST), and occasionally cut whole phrases or sentences.

To investigate time characteristics, we separate the notion of wait time from work time. The vast majority of Shortn's running time is currently spent waiting, because it can take minutes or hours for Turkers to find and accept the task. While wait time is important given the current Mechanical Turk, it is important to remember that the service will continue to grow. Assuming that the number of work tasks does not increase equivalently, wait times will drop. So, while our current median total wait time summed across the three stages was 18.5 minutes (1st Quartile $Q_1 = 8.3$ minutes, 3rd Quartile $Q_3 = 41.6$ minutes), we believe that in the future the worker population will be large enough to consume any task as soon as it is posted.

Considering only work time and assuming negligible wait time, Shortn produced cuts within minutes. We estimate overall work time by examining the median amount of time a worker spent in each stage of the Find-Fix-Verify process. This process reveals that the median shortening took 118 seconds of work time, or just under two minutes,

Input	Original Length	Final Length	Turk Statistics	Time per Paragraph	Example Output
Blog	3 paragraphs 12 sentences 272 words	83% character length	$4.57 158 workers	46 – 57 min	Print publishers are in a tizzy over Apple's new iPad because they hope to ~~finally~~ be able to charge for their digital editions. But in order to get people to pay for their magazine and newspaper apps, they ~~are going to~~ have to offer something different that readers cannot get at the newsstand or on the open Web.
Classic UIST [28]	7 paragraphs 22 sentences 478 words	87%	$7.45 264 workers	49 – 84 min	The metaDESK effort is part of the larger Tangible Bits project~~. The Tangible Bits vision paper~~, which introduced the metaDESK ~~along with~~and two companion platforms, the transBOARD and ambientROOM.
Draft UIST [29]	5 paragraphs 23 sentences 652 words	90%	$7.47 284 workers	52 – 72 min	~~In this paper we argue that~~ it is possible and desirable to combine the easy input affordances of text with the powerful retrieval and visualization capabilities of graphical applications. We present WenSo, ~~a tool that~~which uses lightweight text input to capture richly structured information for later retrieval and navigation in a graphical environment.
Rambling E-mail	6 paragraphs 24 sentences 406 words	78%	$9.72 362 workers	44 – 52 min	~~A previous board member,~~Steve Burleigh~~,~~ created our web site last year and gave me alot of ideas. ~~For this year,~~I found a web site called eTeamZ that hosts web sites for sports groups. Check out our new page: […]
Technical Comp. Sci. [3]	3 paragraphs 13 sentences 291 words	82%	$4.84 188 workers	132 – 489 min	Figure 3 shows the pseudocode that implements this design for Lookup. FAWN-DS extracts two fields from the 160-bit key: ~~the i low order bits of the key~~ (the index bits) and the next 15 low order bits ~~(the key fragment)~~.

Table I. Our evaluation run of Shortn produced revisions between 78% – 90% of the original paragraph length on a single run. The Example Output column contains example edits from each input.

when summed across all three stages (Q_1 = 60 seconds, Q_3 = 3.6 minutes). As Mechanical Turk grows, users may see shortening tasks approaching a limit of two minutes.

The average paragraph cost $1.41 to shorten under our pay model. This cost split into $0.55 to identify an average of two patches, then $0.48 to generate alternatives and $0.38 to filter results for each of those patches. Were we instead to use a $0.01 pay rate for these tasks, the process would cost $0.30 per paragraph. Our experience is that paying less slows down the later parts of the process, but it does not impact quality [19] — it would be viable for shortening paragraphs under a loose deadline.

Qualitatively, Shortn was most successful when the input had unnecessary text. For example, with the Blog input, Shortn was able to remove several words and phrases without changing the meaning of the sentence. Workers were able to blend these cuts into the sentence easily. Even the most technical input texts had extraneous phrases, so Shortn was usually able to make at least one small edit of this nature in each paragraph.

Shortn occasionally introduced errors into the paragraph. While Turkers tended to stay away from cutting material they did not understand, they still occasionally flagged such patches. As a result, Turkers sometimes made edits that were grammatically appropriate but stylistically incorrect. For example, it may be inappropriate to remove the academic signaling phrase "In this paper we argue that…" from an introduction. Cuts were a second source of error: Turkers in the Fix stage would vote that a patch could be removed entirely from the sentence, but were not given the chance to massage the cut into the sentence. So, cuts often led to capitalization and punctuation problems at sentence boundaries. Modern auto-correction techniques could catch many of these errors. Parallelism was another source of

error: for example, in Technical Computer science (Table I), the two cuts were from two different patches, and thus handled by separate Turkers. These Turkers could not predict that their cuts would not match, one cutting the parenthetical and the other cutting the main phrase.

To investigate the extent of these issues, we coded all 126 shortening suggestions as to whether they led to a grammatical error. 37 suggestions were ungrammatical, again supporting our rule of thumb that 30% of raw Turker edits will be noisy. The Verify step caught 19 of the errors (50% of 37) while also removing 15 grammatical sentences: its error rate was thus (18 false negatives + 15 false positives) / 137 = 26.1%, again near 30%. Microsoft Word's grammar checker caught 13 of the errors. Combining Word and Shortn caught 24 of the 37 errors.

We experimented with feeding the shortest output from the Blog text back into the algorithm to see if it could continue shortening. It continued to produce cuts between 70–80% with each iteration. We ceased after 3 iterations, having shortened the text to less than 50% length without sacrificing readability or major content. The user can take advantage of this functionality by pushing the Shortn button again once the results come back.

Crowdproof Evaluation

To evaluate Crowdproof, we obtained a set of five input texts in need of proofreading (Table II). We manually labeled all spelling, grammatical and style errors in each of the five inputs, identifying a total of 49 errors. We then ran Crowdproof on the inputs using a 20-minute stage timeout, with prices $0.06 for Find, $0.08 for Fix, and $0.04 for Verify. We measured the errors that Crowdproof caught, that Crowdproof fixed, and that Word caught. We ruled that Crowdproof had caught an error if one of the identified patches contained the error.

Input	Content	Errors all/caught/fixed	Turkers	Time	Example Output
Passes Word's Checker [4]	1 paragraph 4 sentences 49 words	9 / 9 / 8	$4.76 77 workers	48 min	Marketing ~~are~~is bad for brands big and small. You ~~K~~know ~~W~~what I am ~~S~~saying. It is no wondering that advertis~~ing~~ements ~~are~~ is bad for compan~~y~~ies in America, Chicago and Germany. Updating of brand image ~~are~~is bad for processes in one company and many companies.
ESL	1 paragraph 8 sentences 166 words	12 / 5 / 4	$2.26 38 workers	47 min	However, while GUI made using computers ~~be~~ more intuitive and easier to learn, it didn't ~~let people be able to~~ control computers efficiently. ~~Massesnis only can~~The masses only can use the software developed by software companies, unless they know how to write programs.
Notes	2 paragraphs 8 sentences 107 words	14 / 8 / 8	$4.72 79 workers	42–53 min	~~Blah blah blah—~~This is an argument about whether there should be a standard "~~nosql~~ NoSQL storage" API to protect developers storing their stuff in proprietary services in the cloud. ~~Probably unrealistic.~~ To protect yourself, use an open software offering, ~~and~~ self-host or go with hosting solution that uses open offering.
Wikipedia	1 paragraph 5 sentences 63 words	8 / 7 / 6	$2.18 36 workers	54 min	Dandu Monara (Flying Peacock, Wooden Peacock), The Flying ~~m~~Machine ~~able to fly~~. The King Ravana (Sri Lanka) built it. Accor~~i~~ding to ~~h~~Hindu believe~~s~~rs in Ramayanaya King Ravana used "Dandu Monara" for abduct queen Seetha from Rama. According to believers, "Dandu Monara" landed at Werangatota.
UIST Draft	1 paragraph 6 sentences 135 words	6 / 4 / 3	$3.30 53 workers	96 min	Many of these problems vanish if we turn to a much older recording technology--text. When we enter text, each (pen or key) stroke is being used to record the actual information we care about; none is wasted on application navigation or configuration.

Table II. A report on Crowdproof's runtime characteristics and example output.

Results

Soylent's proofreading algorithm caught 33 of the 49 errors (67%). For comparison, Microsoft Word's grammar checker found 15 errors (30%). Combined, Word and Soylent flagged 82% of all errors. Word and Soylent tended to identify different errors, rather than both focusing on the easy and obvious mistakes. This result lends more support to Crowdproof's approach: it will waste relatively little money that an AI could have saved.

Crowdproof was effective at fixing errors that it found. Using the Verify stage to choose the best textual replacement, Soylent fixed 29 of the 33 errors it flagged (88%). To investigate the impact of the Verify stage, we labeled each unique correction that Turkers suggested as grammatical or not. Fully 28 of 62 suggestions, or 45%, were ungrammatical. The fact that such noisy suggestions produced correct replacements again suggests that Turkers are much better at verification than they are at authoring.

Crowdproof's most common problem was missing a minor error that was in in the same patch as a more egregious error. The four errors that Crowdproof failed to fix were all contained in patches with at least one other error; Lazy Turkers fixed only the most noticeable problem. A second problem was a lack of domain knowledge: in the ESL example in Table II, Turkers did not know what a GUI was, so they could not know that the author intended "GUIs" instead of "GUI". There were also stylistic opinions that the original author might not have agreed with: in the Draft UIST example in Table II, the author clearly had a penchant for triple dashes that the Turkers did not appreciate.

Crowdproof shared many running characteristics with Shortn. Its median work time was 2.8 minutes (Q_1 = 1.7 minutes, Q_3 = 4.7 minutes), so it completes in very little work time. Similarly to Shortn, its wait time was 18 minutes (Median = 17.6, Q_1 = 9.8, Q_3 = 30.8). It cost more money to run per paragraph (μ=$3.40, σ=$2.13) because it identified far more patches per paragraph: we chose paragraphs in dire need of proofreading.

Human Macro Evaluation

We were interested in understanding whether end users could instruct Mechanical Turk workers to perform open-ended tasks. Can users communicate their intention clearly? Can Turkers execute the amateur-authored tasks correctly?

Method

We generated five feasible Human Macro scenarios (Table III). We recruited two sets of users: five undergraduate and graduate students in our computer science department (4 male) and five administrative associates in our department (all female). We showed each user one of the five prompts, consisting of an example input and output pair. We purposefully did not describe the task to the participants so that we would not influence how they wrote their task descriptions. We then introduced participants to The Human Macro and described what it would do. We asked them to write a task description for their prompt using The Human Macro. We then sent the description to Mechanical Turk and requested that five Turkers complete each request. In addition to the ten requests generated by our participants, one author generated five requests himself to simulate a user who is familiar with Mechanical Turk.

We coded results using two quality metrics: intention (did the Turker understand the prompt and make a good faith effort?) and accuracy (was the result flawless?). If the Turker completed the task but made a small error, the result was coded as good intention and poor accuracy.

Results

Users were generally successful at communicating their intention (Table III). The average command saw an 88% intention success rate (max = 100%, min = 60%). Typical intention errors occurred when the prompt contained two requirements: for example, the Figure task asked both for an image and proof that the image is Creative Commons-

Task	Quality	Example Request	Example Input	Example Output
Tense $0.10 1 paragraph	CS: 100% intention, (20% accuracy) Admin: 100% (40%) Author: 100% (60%)	Admin: "Please change text in document from past tense to present tense."	I gave one final glance around before descending from the barrow. As I did so, my eye caught something […]	I give one final glance around before descending from the barrow. As I do so, my eye catches something […]
Figure $0.20 1 paragraph	CS: 75% (75%) Admin: 75% (75%) Author: 60% (60%)	CS: "Pick out keywords from the paragrah like Yosemite, rock, half dome, park. Go to a site which hsa CC licensed images […]"	When I first visited Yosemite State Park in California, I was a boy. I was amazed by how big everything was […]	http://commons.wikimedia.org/wiki/File:03_yosemite_half_dome.jpg
Opinions $0.15 1 paragraph	CS: 100% (100%) Admin: 100% (100%) Author: 100% (100%)	CS: "Please tell me how to make this paragraph communicate better. Say what's wrong, and what I can improve. Thanks!"	Take a look at your computer. Think about how you launch programs, edit documents, and browse the web. Don't you feel a bit lonely? […]	This paragraph needs an objective I feel like. […] After reading I feel like there should be about five more sentences […]
Citation Gathering $0.40 3 citations	CS: 75% (75%) Admin: 100% (100%) Author: 66% (40%)	Admin: "Hi, please find the bibtex references for the 3 papers in brackets. You can located these by Google Scholar searches and clicking on bibtex."	Duncan and Watts [Duncan and watts HCOMP 09 anchoring] found that Turkers will do more work when you pay more, but that the quality is no higher."]	@conference{ title={{Financial incentives and […]}}, author={Mason, W. and Watts, D.J.}, booktitle={HCOMP '09}}
List Processing $0.05 10 inputs	CS: 82% (82%) Admin: 98% (96%) Author: 91% (68%)	Admin: "Please complete the addresses below to include all informtion needed as in example below. […]"	Max Marcus, 3416 colfax ave east, 80206	Max Marcus 3416 E Colfax Ave Denver, CO 80206

Table III. The five tasks in the left column led to a variety of request strategies. Terse, error-filled user requests still often led to success.

licensed. Turkers read far enough to understand that they needed to find a picture, found one, and left. Successful users clearly signaled Creative Commons status in the title field of their request.

With accuracy, we again see that roughly 30% of work contained an error. (The average accuracy was 70.8%.) Turkers commonly got the task mostly correct, but failed on some detail. For example, in the Tense task, some Turkers changed all but one of the verbs to present tense, and in the List Processing task, sometimes a field would not be correctly capitalized or an Eager Beaver would add too much extra information. These kinds of errors would be dangerous to expose to the user, because the user might likewise not realize that there is a small error in the work.

DISCUSSION

This section reviews some fundamental questions about the nature of paid, crowd-powered interfaces as embodied in Soylent. Our work suggests that it may be possible to transition from an era where Wizard of Oz techniques were used only as prototyping tools to an era where a "Wizard of Turk" can be permanently wired into a system. We touch on resulting issues of wait time, cost, legal ownership, privacy, and domain knowledge.

In our vision of interface outsourcing, authors have immediate access to a pool of human expertise. Lag times in our current implementation are still on the order of minutes to hours, due to worker demographics, worker availability, the relative attractiveness of our tasks, and so on. While future growth in crowdsourced work will likely shorten lag times, this is an important avenue of future work. It may be possible to explicitly engineer for responsiveness in return for higher monetary investment, or to keep workers on retainer with distractor tasks until needed [3].

With respect to cost, Soylent requires that authors pay all workers for document editing — even if many changes never find their way into the final work product. One might therefore argue that interface outsourcing is too expensive to be practical. We counter that in fact all current document processing tasks also incur significant cost (in terms of computing infrastructure, time, software and salaries); the only difference is that interface outsourcing precisely quantifies the price of each small unit of work. While payment-per-edit may restrict deployment to commercial contexts, it remains an open question whether the gains in productivity for the author are justified by the expense.

Regarding privacy, Soylent exposes the author's document to third party workers without knowing the workers' identities. Authors and their employers may not want such exposure if the document's content is confidential or otherwise sensitive. One solution is to restrict the set of workers that can perform tasks: for example, large companies could maintain internal worker pools. Rather than a binary opposition, a continuum of privacy and exposure options exists.

Soylent also raises questions over legal ownership of the resulting text, which is part-user and part-Turker generated. Do the Turkers who participate in Find-Fix-Verify gain any legal rights to the document? We believe not: the Mechanical Turk worker contract explicitly states that it is work-for-hire, so results belong to the requester. Likewise with historical precedent: traditional copyeditors do not own their edits to an article. However, crowdsourced interfaces will need to consider legal questions carefully.

A final concern is that anonymous workers may not have the necessary domain knowledge or enough shared context to usefully contribute. We agree that some tasks, like fleshing out a related work section in an academic paper based on bullet points, are much more difficult to achieve on to-

day's Mechanical Turk. However, a large subset of editing tasks only requires generic editing skills. We also may effectively personalize by directing tasks to Turkers who have successfully worked on a user's documents before.

CONCLUSION

The following conclusion was Shortn'ed to 85% length:

This paper presents Soylent, a word processing interface that uses crowd workers to help with proofreading, document shortening, editing and commenting tasks. Soylent is ~~an example of a~~ new ~~kind of~~ interactive user interface in which the end user has direct access to a crowd of workers for assistance with tasks that require human attention and common sense. Implementing these ~~kinds of~~ interfaces requires new software programming patterns ~~for interface software~~, since crowds behave differently than computer systems. We have introduced one important pattern, Find-Fix-Verify, which splits complex editing tasks into a series of identification, generation, and verification stages ~~that use independent agreement and voting~~ to produce reliable results. We evaluated Soylent with a range of editing tasks, finding and correcting 82% of grammar errors ~~when combined with automatic checking~~, shortening text to approximately 85% of original length per iteration, and executing a variety of human macros successfully.

Future work falls in three categories. First are new crowd-driven features for word processing, such as readability analysis, smart find-and-replace ~~(so that renaming "Michael" to "Michelle" also changes "he" to "she")~~, and figure or citation number checking. Second are new techniques for optimizing crowd-programmed algorithms to reduce wait time and cost. Finally, we believe that our research points the way toward integrating on-demand crowd work into other authoring interfaces, particularly in creative domains ~~like image editing and programming~~.

ACKNOWLEDGMENTS

REFERENCES

1. Bernstein, M., Marcus, A., Karger, D.R., and Miller, R.C. Enhancing Directed Content Sharing on the Web. *CHI '10*, ACM Press (2010).

2. Bernstein, M., Tan, D., Smith, G., Czerwinski, M., et al. Collabio: A Game for Annotating People within Social Networks. *UIST '09*, ACM Press (2009), 177–180.

3. Bigham, J.P., Jayant, C., Ji, H., Little, G., et al. VizWiz: Nearly Real-time Answers to Visual Questions. *UIST '10*, ACM Press (2010).

4. Clarke, J. and Lapata, M. Models for sentence compression: a comparison across domains, training requirements and evaluation measures. *ACL '06*, Association for Computational Linguistics (2006).

5. Cohn, T. and Lapata, M. Sentence compression beyond word deletion. *COLING '08*, (2008).

6. Cypher, A. *Watch What I Do*. MIT Press, Cambridge, MA, 1993.

7. Dourish, P. and Bellotti, V. Awareness and coordination in shared workspaces. *CSCW '92*, ACM Press (1992).

8. Evans, B. and Chi, E. Towards a model of understanding social search. *CSCW '08*, ACM Press (2008).

9. Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S. What Would Other Programmers Do? Suggesting Solutions to Error Messages. *CHI '10*, ACM Press (2010).

10. Heer, J. and Bostock, M. Crowdsourcing Graphical Perception: Using Mechanical Turk to Assess Visualization Design. *CHI '10*, ACM Press (2010).

11. Kittur, A., Chi, E.H., and Suh, B. Crowdsourcing user studies with Mechanical Turk. *CHI '08*, ACM Press (2008).

12. Knight, K. and Marcu, D. Summarization beyond sentence extraction: a probabilistic approach to sentence compression. *Artificial Intelligence 139*, 1 (2002).

13. Krieger, M., Stark, E.M., and Klemmer, S.R. Coordinating tasks on the commons: designing for personal goals, expertise and serendipity. *CHI '09*, ACM Press (2009).

14. Kukich, K. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR) 24*, 4 (1992).

15. Lieberman, H. *Your Wish is My Command*. Morgan Kaufmann, San Francisco, 2001.

16. Little, G., Chilton, L., Goldman, M., and Miller, R.C. TurKit: Human Computation Algorithms on Mechanical Turk. *UIST '10*, ACM Press (2010).

17. Little, G., Lau, T.A., Cypher, A., Lin, J., et al. Koala: Capture, Share, Automate, Personalize Business Processes on the Web. *CHI '07*, (2007).

18. Marcu, D. *The Theory and Practice of Discourse Parsing and Summarization*. MIT Press, 2000.

19. Mason, W. and Watts, D. Financial Incentives and the "Performance of Crowds". *ACM SIGKDD Workshop on Human Computation*, ACM Press (2009).

20. Miller, R. and Myers, B. Interactive simultaneous editing of multiple text regions. *USENIX '01*, (2001).

21. Quinn, A.J. and Bederson, B.B. A Taxonomy of Distributed Human Computation.

22. Ross, J., Irani, L., Silberman, M.S., Zaldivar, A., et al. Who Are the Crowdworkers? Shifting Demographics in Amazon Mechanical Turk. *alt.chi '10*, ACM Press.

23. Sala, M., Partridge, K., Jacobson, L., and Begole, J. An Exploration into Activity-Informed Physical Advertising Using PEST. *Pervasive '07*, Springer Berlin Heidelberg (2007).

24. Simon, I., Morris, D., and Basu, S. MySong: automatic accompaniment generation for vocal melodies. *Proc. CHI '08*, ACM Press (2008).

25. Snow, R., O'Connor, B., Jurafsky, D., and Ng, A.Y. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. *ACL '08*, (2008).

26. Sorokin, A. and Forsyth, D. Utility data annotation with Amazon Mechanical Turk. *CVPR '08*, (2008).

27. von Ahn, L. and Dabbish, L. Labeling images with a computer game. *CHI '04*, ACM Press (2004).