Arguing a Research Project

CS 197 | Stanford University | Alex Tamkin
cs197.stanford.edu/01
Seating Shuffle

Organize yourself into a group of 3-4

Ensure your group has a few people outside of your project group!

Meet new folks & introduce yourself!
You should all have projects/groups. Let us know if that’s not the case.

Assignment 3 — Project Introduction — is out later today and due next Wednesday.

After Assignment 3, your main goal is to make self-guided progress on the project through the rest of the quarter! We will provide scaffolds via assignment check-ins.

Heads up on Assignment 2 — Related Work:

After we grade this weekend, you’ll come in to your TA’s OH (Week 4) for an interactive-grading session to address feedback and fixes for final paper.
Administrivia

Slack

AI tutorial - virtual (shared)

Copying text and attribution

Publishing and project authorship
Last time

How do we get to the point where we know what has been done, and why our idea is different, new, and exciting?

Bit flip: articulating an assumption present in all prior work that you are breaking

Literature search process:

Iterative expansion of the most relevant work from the set of papers you've seen so far
Today: from bit flip to paper introduction

How do we articulate our project persuasively to a peer? A bit flip isn’t enough on its own.

If we can’t explain the project clearly enough for another researcher in the same area to understand it, we don’t really understand our project ourselves.

(This happens more often than you might think…It’s hard!)
Crowdsourcing mobilizes a massive online workforce into collectives of unprecedented scale. The dominant approach for crowdsourcing is the microtask workflow, which enables contributions at scale by modularizing and pre-specifying all actions [7, 55]. By drawing together experts [70] or amateurs [6], microtask workflows have produced remarkable success in robotic control [48], data clustering [12], galaxy labeling [54], and other goals that can be similarly pre-specified. However, goals that are open-ended and complex, for example invention, production, and engineering [42], remain largely out of reach. Open-ended and complex goals are not easily adapted to microtask workflows because it is difficult to articulate, modularize, and pre-specify all possible actions needed to achieve them [71, 80]. If crowdsourcing remains confined to only the goals so predictable that they can be entirely pre-defined using workflows, crowdsourcing’s long-term applicability, scope and value will be severely limited.

In this paper, we explore an alternative crowdsourcing approach that can achieve far more open-ended and complex goals: crowds structured like organizations. We take inspiration from modern organizations because they regularly orchestrate large groups in pursuit of complex and open-ended goals, whether short-term like disaster response or long-term like spaceflight [8, 9, 63]. Organizations achieve this complexity through a set of formal structures — roles, teams, and hierarchies — that encode responsibilities, interdependencies and information flow without necessarily pre-specifying all actions [15, 83].

We combine organizational structures with computational crowdsourcing techniques to create flash organizations, rapidly assembled and reconﬁgurable organizations composed of online crowd workers (Figure 1). We instantiated this approach in a crowdsourcing platform that computationally convenes large groups of expert crowd workers and directs their efforts to achieve complex goals such as product design, software development and game production.

We introduce two technical contributions that address the central challenges in structuring crowds like organizations. The first problem: organizations typically assume asset specificity, the ability for organization members to develop effective collaboration patterns by working together over time [83]. Clearly, crowds, with workers rapidly assembled on-demand from platforms such as Upwork (www.upwork.com), do not offer asset specificity. So, our system encodes the division of labor into a de-individualized role hierarchy, inspired by movie crews [2] and disaster response teams [8], enabling workers to coordinate using their knowledge of the roles rather than their knowledge of each other.

The second problem: organizational structures need to be continuously reconfigured so that the organization can adapt as work progresses, for example by changing roles or adding teams [9, 63, 83]. Coordinating many workers' reconfigurations in parallel, however, can be challenging. So, our system enables reconfiguration through a model inspired by version control: workers replicate (branch) the current organizational structure and then propose changes (pull requests) for those up the hierarchy chain to review, including the addition of new tasks or roles, changes to task requirements, and revisions of the organizational hierarchy itself.

Enabling new forms of organization could have dramatic impact: organizations have become so influential as the backbone of modern economies that Weber argued them to be the most important social phenomenon of the twentieth century [82]. Flash organizations advance a future where organizations are no-longer anchored in traditional Industrial Revolution-era labor models, but are instead fluidly assembled and re-assembled from globally networked labor markets. These properties could eventually enable organizations to adapt at greater speed than today and prototype new ideas far more quickly.

In the rest of the paper, we survey the foundations for this work and describe flash organizations and their system infrastructure. Following this review, we present an evaluation of three flash organizations and demonstrate that our system allows crowds, for the first time, to work iteratively and adaptively to achieve complex and open-ended goals. The three organizations used our system to engage in complex collective behaviors such as spinning up new teams quickly when unplanned changes arose, training experts on-demand in areas such as medical privacy policy when the crowd marketplace could not provide the expertise, and enabling workers to suggest bottom-up changes to the work and the organization.
Architecture of an Introduction
What is an Introduction?

The Introduction makes the case for your research, in brief.

Jennifer Widom, Dean of Engineering:

“The Introduction is crucially important. By the time a referee has finished the Introduction, they've probably made an initial decision about whether to accept or reject the paper — they'll read the rest of the paper looking for evidence to support their decision.

A casual reader will continue on if the Introduction captivated them, and will set the paper aside otherwise. Again, the Introduction is crucially important.”

https://cs.stanford.edu/people/widom/paper-writing.html#intro
What is an Introduction?

The Introduction makes the case for your research, in brief.

Jennifer Widom, Dean of Engineering:

“The Introduction is crucially important. By the time a referee has finished the Introduction, they've probably made an initial decision about whether to accept or reject the paper — they'll read the rest of the paper looking for evidence to support their decision. A casual reader will continue on if the Introduction captivated them, and will set the paper aside otherwise. Again, the Introduction is crucially important.”

https://cs.stanford.edu/people/widom/paper-writing.html#intro

Bookmark this advice page! Revisit when you submit your first publication!
Think of it this way...

By this point, the video has hopefully made clear to you what it’s about, and you’ve made a decision about whether to watch the rest of it.
Each introduction makes the case for two things:

1) The **problem**: why do we care about the problem you’re solving?
2) The **solution**: why is your approach creative and correct?
Architecture of an intro

Problem
Solution

...great, Alex, thanks. But how do we actually do this?
The Problem

By yourself, jot down an explanation of the problem that your project is working on.

How clearly do you understand your own problem?

How clearly do you understand your own bit flip?

(Maybe not super well at this point! Deep understanding the problem often requires a lot of context / immersion)
Unpacking the problem

The Introduction’s goal isn’t just to set up the problem, it’s to convey the solution as well. To do that effectively, your problem statement needs to set up the bit flip.

For this to succeed, the bit needs to integrated as part of the problem statement.
Problem motivation

Explain the main problem that you’re trying to solve:

- Networks are hard to (re)configure
- Interactions with computers are stuck on flat glass displays
- Generative AI models are challenging to evaluate

Use citations to back up your claims about the existence of the problem, and why we should care about solving it.

Problem motivation
Set up the bit
Solution (bit flip)
Set up the bit

Answer the question, "Why isn't this problem solved yet?" by setting up the bit that you're going to flip:

Networks are configured in hardware

To break out of glass screens, outputs have been designed into the physical world.

Generative model evaluations have been automated, but these are proxies at best.

This is a summary of related work that is in service of your bit set up.

Problem motivation

Set up the bit

Solution (bit flip)
Crowdsourcing platforms such as Amazon Mechanical Turk decentralize their workforce, designing for distributed, independent work [16, 42]. Decentralization aims to encourage accuracy through independent judgement [59]. However, by making communication and coordination more difficult, decentralization disempowers workers and forces worker collectives off-platform [41, 64, 16]. The result is disenfranchise-ment [22, 55] and an unfavorable workplace environment [41, 42]. Worse, while decentralization is motivated by a desire for high-quality work, it paradoxically undercuts behaviors and institutions that are critical to high-quality work. In many traditional organizations, for example, centralized worker coordination is a keystone to behaviors that improve work quality, including skill development [2], knowledge management [35], and performance ratings [58].

Your Introduction cites essential work for setting up the bit. Exhaustive lit review goes in your Related Work.
Try again: The Problem

In your group, explain the problem that your project is working on [1 min each]

How clearly do you understand your partner’s problem?

How clearly do you understand your partner’s bit flip?

Problem motivation
Set up the bit
Solution (bit flip)

(a total of 3-4 min here)
Architecture of an intro

Problem statement
Set up the bit
Solution (bit flip)
The Solution

By yourself, jot down the approach your project is taking.

How clearly do you understand your own bit flip?

How clearly do you understand how exactly the project is going to instantiate that bit flip in a specific system, algorithm, or design?
Unpacking the solution

The solution has to explain two things: what the big idea is, and how that big idea gets instantiated in the specific context of this problem.

(Even if someone hears your bit flip that you want to introduce recurrence inside the neural network, they may still have no idea how that actually connects to the problem of language generation.)
Flip the bit

The topic sentence of this paragraph is the thesis statement of your entire research project.

Pivot off of the bit you set up to flip the bit. Explain why flipping the bit is a good idea for the problem at hand.

It should now be obvious to a reader given the prior paragraph that this research is novel, since you have proven that nobody else has flipped that bit.
To address this reputation challenge, and with an eye toward other challenges that arise from decentralization, we draw inspiration from a historical labor strategy for coordinating a decentralized workforce: *guilds*. Worker guilds arose in the early Middle Ages, when workers in a trade such as silk were distributed across a large region, as bounded sets of laborers who shared an affiliation. These guilds played many roles, including training apprentices [18, 44], setting prices [45], and providing mechanisms for collective action [52, 49]. Especially relevant to the current challenge, guilds measured and certified their own members’ quality [18]. While guilds eventually lost influence due to exerting overly tight controls on trade [45] and exogenous technical innovations in production, their intellectual successors persist today as professional organizations such as in engineering, acting and medicine [46, 33]. Malone first promoted a vision of online “e-lancer” guilds twenty years ago [40], but to date no concrete instantiations exist for a modern, online crowd work economy.
At this point, the reader understands the idea that you're proposing, but it's still very high level. In this paragraph, map that idea onto a concrete instantiation.

Typically, this is where the system or algorithm that you're creating gets a name. Explain its architecture or design at a high level. Make clear how this architecture or design is an instance of the bit flip.
We present *crowd guilds*: crowd worker collectives that coordinate to certify their own members and perform internal feedback to train members (Figure 1). Our infrastructure for crowd guilds enables workers to engage in continuous double-blind peer assessment [30] of a random sample of members’ task submissions on the crowdsourcing platform, rating the quality of the submission and providing critiques for further improvement. These peer assessments are used to derive guild levels (e.g., Level 1, Level 2) to serve as reputation (qualification) signals on the crowdsourcing platform. As workers gather positive assessments from more senior guild members, they rise in levels within the guild. Guilds translate these levels into higher wages by recommending pay rates for each level when tasks are posted to the platform. While crowd guilds focus here on worker reputation, our experiment implementation also explores how crowd guilds could address other challenges such as collective action (e.g., collectively rejecting tasks that pay too little), formal mentorship (e.g., repeated feedback and training), and social support (e.g., on the forums). Because
Try again: The Solution

In your room, explain the approach your project is taking (or might take!) [1 min each]

How clearly do you understand your partner’s bit flip?

How clearly do you understand how exactly the project is going to instantiate that bit flip in a piece of software?

(a total of 3-4 min here)

Problem motivation
Set up the bit
Flip the bit
Instantiate the bit flip
Evaluation

How did you prove that your bit flip is successful at solving the problem?

We obviously haven’t covered evaluation yet in this course, so for now you’ll need to take your best guess.

How would you convince a critical reader that flipping the bit solved the problem better than the prior work?
Implications

If you’re right and the bit flip is how everyone should be approaching this problem from now on, what implications are there for the field?

This is your chance to stand on a small soapbox:

- Will it change the contexts in which we use this technology? Will it broaden usage?

But don’t overplay your hand:

- It probably won’t change all of computing.
So in brief: use your literature search to motivate your problem and set up a bit. Then, flip the bit and argue persuasively that this will address the problem. Explain how this solution gets built into your system or model.
How to Write
The Introduction
First, find your genre

There are a few different kinds of paper that are common:

- New problem / old solution
- Old problem / new solution
Activity recognition (new) solved with off-the-shelf ML (old)

State of the literature

Address a new problem with an old solution

Address a new problem with a new solution

Hard to convince the world

Question answering (old) with a transformer architecture (new)

Address an old problem with a new solution
Answer a new question with an old method

Solve a new problem with a new technique

Social media disclosures of mental illness

State of the literature

Tie strength and Facebook use

Hard to convince the world

Answer an old question with a new method
Why only make one move?

When making an argument, you want to introduce one major new idea, to minimize the new ideas your listener needs to absorb.

Certain ideas already have warrants in the literature: prior work already has proven their legitimacy. A warrant is a free pass!

Old problem: the problem already has a warrant in the literature.

Visual question answering is a legitimate task; mission critical code should be proven correct; interaction should not happen on panes of glass

Old solution: the solution already has a warrant in the literature.

Sensor fusion into features for an ML system; transformer architectures for NLP; tangible interaction; self-play in reinforcement learning
Why only make one move?

Typically you are spending the introduction making the case for your new idea. If you are trying to make the case for both a new problem and a new solution, a reader might disagree with either. This is not to say that you can’t do new problem / new solution; just that it’s a risky varsity maneuver.
From genre to intro

Old problem / new solution:
- Motivate the problem via prior work, which has already established the problem
- Set up the bit of how all prior work tried to solve it
- Flip the bit — your new solution
- Instantiate that new solution
- Implications

New problem / old solution:
- Motivate the problem via rhetoric, drawing on prior work making supporting claims
- Set up the bit: prior work is not equipped for this problem
- Flip the bit — your new solution
- Instantiate that new solution
- Implications
Start with an outline

Your idea should be fully understandable with only six sentences, a topic sentence per paragraph:

- Problem motivation
- Set up the bit
- Flip the bit
- Instantiate the bit flip
- Evaluation
- Implications
Keep it taut

Your goal is then to treat each outline point as a thesis sentence for the paragraph, and use the paragraph to prove that thesis. Don’t stray and make other interesting but un-useful points.
Example 1

This is an early draft of a paper that Lisa submitted (and got rejected 😞).

Ignoring the Casual English, the reader is essentially walking in the dark:

- New problem with zero prior context
- 2+ ideas introduced in each ¶
- No clear flip (too many bits/un-useful points)
- No idea of how solution will help (no evaluation, no implication)

INTRODUCTION

Classes are getting large. There’s a large focus on graphics-based programming assignments. These are great for increasing engagement in the topic of computer science, as they make the assignments more relatable to students, or more applicable to various interdisciplinary activities (cite). But while these assignments can be fun and great for students, it’s really painful to grade.

Something about the increased reliance on student assignments for feedback and understanding. We are moving away from the one-on-one tutoring scenario, and we need to use our human resources well. It is not effective for a student teacher to spend all of their time grading, but the understanding that comes from giving detailed feedback to a student is a key component of their learning experience.

Introduce the vocabulary of snapshot and trajectory.

There are often two facets to grading assignments: style and functionality. There are therefore two options that we can try. We could imagine finely tuned unit tests, which tend to work well on short programs. But as soon as these programs introduce open-ended extra credit, unit tests often fall through. The second option is to analyze the code structure via some abstract syntax tree (AST) or tokenizer approach. Program synthesis and other techniques can be used here. However, this tends to not be great either, because once programs get longer and allow students to use variables with no set naming scheme, it becomes impossible to analyze.

And so therein rises a question. How do we tackle these two challenges at once? If it is difficult to grade functionality by analyzing the student code structure, and it is also difficult to grade functionality, is there a way for us to leverage recent technological advances to assess functionality? Is there also a way to leverage the large numbers of our classrooms to help us? Or even just the huge amount of data available to us?

The answer is yes. The field of computer vision has been advancing rapidly in the last few years to tackle many problems, and they have almost all been solved with deep learning techniques.

In this paper, we present AETHER (Auto-Encoding Trajectories and Helpful Evaluation of Rubrics).

This paper is split into two main components. The first is a discussion of the deep learning algorithms that are used to extract information from the code. The second is a discussion of how these algorithms are used to evaluate student work.
INTRODUCTION
First-time CS students learn by programming—they must design an approach, debug their code, and iteratively improve towards a final solution. For a teacher, however, a single timestamped submission per student at the end of this process is insufficient to capture all the intermediate steps a student has taken towards a solution. Even when such progress data is available, it is often intractable to analyze this data in a meaningful way that maps student code to milestones, or incremental attempts towards the assignment goal. If teachers could characterize individual and aggregate progress through an assignment, they could better understand how their students think about course concepts.

However, student progress is highly variable. Unit testing and other automated assessment tools are often only designed to test functionality of the final submission, and teachers would have to design and engineer additional tests for identifying intermediate milestones. Existing tools are also difficult to tailor to graphics-based programming assignments, which have recently gained popularity in many CS1 courses. Due to their open-ended nature—enabling programmers of all levels to get quick, visual feedback in an exploratory environment—the large solution spaces often render unit-testing development or syntax-based code analysis insufficient for final submissions, much less intermediate code snapshots.

Despite the up-front complexity of autonomously analyzing functionality of graphics-based coding tasks, advances in computer vision in other fields have matched [8] and, more recently, surpassed [14] human ability to detect objects from pixel input. In spite of their effectiveness, contemporary vision classification techniques have rarely been applied to student code, as many state-of-the-art techniques are supervised—requiring well-labeled, plentiful data, which student assignment data often lacks.

In this work, we marry the two fields of understanding students and improving computer vision by presenting the community with a computer vision challenge rooted in CS education: autonomously characterize student progress on a graphics-based programming task by looking at intermediate image output. The PyramidSnapshot dataset is a large, annotated set of images, each tagged with one of 16 milestone labels mapping to functional progress towards the assignment goals of Pyramid, a canonical CS1 graphics-based task. The dataset contains timestamped, program image output of 2633 students over 26 CS1 offerings from the same university, corresponding to 101,636 images of intermediate work, of which 84,127 are annotated with milestones.

Our work is a first step towards a deeper understanding of how students work through graphics-based assignments. We hope that our publication of the PyramidSnapshot dataset is a useful case study on how to prepare graphics-based student data for quantitative functional analysis, as well as how to glean pedagogical insights from such fine-grained data. In this work we describe our approach to labeling a complex dataset by hand, and we show that a neural network-based classifier can extend human labeling effort reasonably well. Finally, we share our insights into how to visualize student progress to identify different student work patterns.

↑ In retrospect, we could have cleared up the introduction with a topic sentence in each paragraph.

Instead, “zoom in” the reader to an uncharted island on a known map.
Assignment 3

Your group writes an Introduction to a paper for your project

Outline the introduction

Turn the outline into text

700-900 words (for groups of 3-4), 600-700 words (for groups of 2)

Due: Next Wednesday  2pm on Canvas

Details at  cs197.stanford.edu/01
Computer Science Research

Slide content shareable under a Creative Commons Attribution-NonCommercial 4.0 International License.