Liquid Democracy for Rating Systems

Hemanath Peddireddi
Department of Computer Science
University of Warwick

Supervised by Markus Brill

14 April 2025

WARWICK
THE UNIVERSITY OF WARWICK

Abstract

TODO: finish
This project explores how liquid democracy can be used to enhance rating systems by integrating it into vodle, an online polling platform where users rate options using sliders. Traditional liquid democracy models do not always reflect participants' true preferences, especially when some users abstain from voting or when some individuals gain too much influence. To address this, the project adds support for ranked delegation and weighted vote splitting.

Keywords:

Contents

Chapter 1

Introduction

1.1 Motivation - TODO (Finalise)

Liquid democracy has strong theoretical appeal as a flexible and participatory decision-making model, but practical implementations remain rare and underexplored. Most existing research assumes idealised conditions, while real-world systems must contend with asynchronous participation, limited user engagement, and the potential for structural issues like delegation cycles or vote loss.

Vodle, as a platform for collective decision-making, presents an opportunity to explore how liquid democracy can be adapted to work in practice. This project is motivated by the need to understand and resolve the challenges of integrating delegation-based voting into an existing system, while improving the platform's fairness, expressiveness, and overall user experience.

1.2 Vodle - will change or remove

Vodle is an online platform where users participate in polls to vote on subjects through user created polls. Each poll contains a set of options, and users provide ratings for each option from 0 to 100, where a larger number means that they prefer the option more, using a slider. When the poll ends, the ratings submitted by voters are then aggregated and a result is calculated.

can insert screenshots

1.3 Liquid Democracy

Liquid democracy is a decision-making system that combines elements of both direct and representative democracy that offers a voter more flexibility than traditional

voting models.

In direct democracy, every participant votes individually on each issue. This model offers the most individual input but can become impractical for large-scale decision-making due to the high level of participation required from each individual. As **?** states, direct democracy assumes that all individuals are both willing and able to engage meaningfully with every decision, which is often not the case in large groups due to the variance in both the interest and knowledge of voters. The cognitive demand of staying informed on all matters, combined with the time commitment necessary for constant participation, makes direct democracy unmanageable at scale.

In a representative democracy, citizens elect officials who make decisions on their behalf for the duration of a fixed term. While this model is scalable and practical for large populations, it introduces several limitations. Elected representatives often make decisions based on party lines, personal convictions, or external influences such as lobbying groups, which may not accurately reflect the preferences of their constituents (**?**). In addition, because elections are infrequent, this system tends to be unresponsive to shifts in public opinion. Citizens are unable to easily adjust or retract their delegation, which limits their ability to influence decisions once representatives are in office (**?**). As a result, participation is both indirect and inflexible, which can lead to disengagement and dissatisfaction among voters.

Liquid democracy addresses these limitations by allowing voters to either cast their votes directly or delegate them to someone that they trust or to abstain from voting entirely (**?**). In comparison to a direct democracy, the bar for participation is lowered as voters no longer need to stay informed and engaged to pass a vote because they can trust a delegate to do it on their behalf. These delegations can also be updated or revoked at any time, giving users more control over how their vote is used in comparison to a traditional representational democracy where your representative can only be changed at certain points in time.

1.4 Project Goal - TODO

The project's main goal is to integrate liquid democracy into the vodle platform.

Key features include ranked delegation and vote splitting ...

1.5 Project Outline - TODO

This report is structured as follows: *will add when report is written*

Chapter 2

Background Research

This chapter provides background context for the development of a liquid democracy system within vodle. It builds on the concepts introduced earlier, focusing on more detailed research into known limitations of liquid democracy and potential solutions proposed in academic literature. Additionally, the technical foundations and design philosophy of vodle as a platform are explored.

Throughout this chapter, several diagrams are used to illustrate how votes move through a liquid democracy system. To clarify the roles of different voters, the following symbols are used:

- **Circle:** *Delegated voter* - a voter that has delegated and therefore does not cast their own vote.
- **Square:** *Casting voter* - a voter that has not delegated and casts their own vote.
- **Triangle:** *Abstaining voter* - a voter that neither delegates nor casts their own vote.

2.1 Liquid Democracy

Liquid democracy, or delegative voting, allows voters to either cast a vote directly, delegate it to someone they trust, or abstain (**?**). A key feature is that delegations are transitive - a chain of users that all delegate to each other sequentially ends with a single final voter who casts their vote on behalf of all those in the chain.

Whilst the transitivity property enables concentration of voting power with trusted individuals, it can also lead to unintended consequences. Chains of delegations may result in cycles that prevent votes from being cast, or allow certain individuals to accumulate an excessive share of influence, creating “super-voters”. These problems amongst others motivate the need for alternative delegation mechanisms, as discussed in the following subsections.

2.1.1 Issues with Liquid Democracy

Delegation cycles

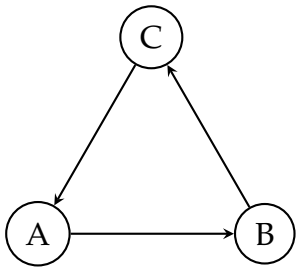

Figure 2.1: Delegation cycle: A delegates to B, B to C, and C back to A.

Delegation cycles occur when a vote is delegated in such a way that it ends up forming a loop (**?**), preventing the vote from reaching a final, resolvable destination. For example, if Alice delegates her vote to Bob, Bob delegates to Charlie, and Charlie delegates back to Alice, the votes become trapped in a cycle (seen above) and can be treated as a loss of representation (**?**).

This issue is particularly problematic because it can nullify votes without the affected users ever realising. In systems where cycles are not explicitly detected and handled, these votes are discarded silently, potentially changing the final outcome of the votes.

A simplistic method to prevent cycles is to check whether a delegation would create a cycle before allowing it. For example, if Alice tries to delegate her vote to Bob, the system checks whether Bob has already directly or indirectly delegated their vote to Alice. If so, the delegation is rejected. However, this approach can be cumbersome and may lead to a poor user experience, as users may not understand why their delegation was denied.

Delegation cycles are increasingly likely to emerge in dynamic voting systems, where delegations can be added, removed, or modified at any point in time. Delegations that initially did not form part of a cycle may later contribute to one as other voters add a new delegation or alter an existing one.

Abstentions

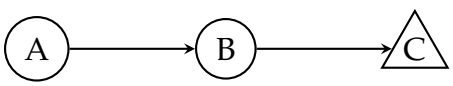

Figure 2.2: Delegation chain ending in abstention: A delegates to B, B to C. C abstains, causing the votes of A and B to be lost.

In liquid democracy, abstention is where a voter neither casts a vote nor delegates their vote to another user (**?**). This includes both deliberate abstention, where a voter knowingly chooses not to participate, and passive abstention, where a voter may be unaware of an ongoing poll or are unable to engage with it.

Abstentions are especially impactful when they occur at the end of a larger delegation chain, as all votes passed along the chain to that voter are effectively lost (**?**). The voters whose decisions were passed along the chain may also be unaware that their votes have been nullified, worsening the effect of the abstention.

Super-voters

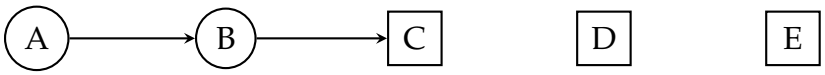

Figure 2.3: Super-voter: A delegates to B, B to C. No matter which vote D or E cast, C's vote will always determine the outcome as it has a weight of 3.

In liquid democracy, a super-voter is an individual who receives a large number of delegated votes, therefore gaining disproportionate influence over decisions (**?**). While this behaviour may reflect voters' genuine preferences, it can lead to a concentration of power that goes against the intended egalitarianism and democratic ideals of liquid democracy.

Although liquid democracy allows users to alter their delegation at any time, in practice, many voters may not actively monitor or even know how their vote is being used. This can allow a small number of super-voters to dominate outcomes, especially in systems with large delegation chains.

Real-world examples of this phenomenon have been documented. In the German Pirate Party's use of LiquidFeedback, certain users received so many delegations that their votes were like "decrees" (**??**) even though they were not elected officials. **?** noted that the super-voters generally voted in line with the majority, therefore not drastically affecting the outcome of the votes and contributed to the stability of the system. However, the potential for individuals to single-handedly influence the results remained a concern.

This pattern is not limited to traditional online voting platforms. It can also be seen within decentralised autonomous organisations (DAOs) - blockchain-based entities where decisions are made collectively by token holders without central leadership. These organisations use token-based voting to decide on critical issues like protocol upgrades and funding allocations. **?** studied 18 decentralised autonomous organisations (DAOs) and found that voting power was often concentrated in the hands of a few delegates. While most did not control a large share of all available tokens, low

participation meant that their share of actual votes cast was disproportionately high. In several DAOs, the top five delegates accounted for over 50% of all votes cast, and in the DAO Gitcoin, this figure exceeded 90%.

2.1.2 Variations of Liquid Democracy

The challenges discussed in the previous section, such as delegation cycles, vote loss due to abstentions, and the emergence of super-voters, highlight inherent vulnerabilities in the standard liquid democracy model. To mitigate these issues, a range of enhancements have been proposed that modify how delegations are expressed, resolved, or overridden. These include techniques that allow voters to specify multiple delegates or distribute their vote to multiple casting voters. Each approach introduces different trade-offs and requires algorithmic support to ensure sound and interpretable outcomes.

The following subsections present several such variations, along with the algorithms that support them.

Ranked Delegation

Ranked delegation improves liquid democracy by allowing voters to list several trusted delegates in order of preference. Instead of choosing just one delegate, a voter can specify a ranked list so that if their top choice is unavailable (due to abstention, involvement in a cycle, or other reasons) the system can use the next delegate specified. This method reduces the risk of losing votes while keeping voters actively involved in the decision making process (**?**).

Implementing ranked delegation requires a mechanism to decide among multiple possible delegation paths. This is done through a *delegation rule*, a function that, given a ranked delegation instance and a delegating voter, selects a unique path leading to a *casting voter* (**?**).

Several key properties help evaluate these delegation rules:

- **Guru Participation:** Ensures that a voter accepting delegated votes (a "guru") is never worse off by doing so. Receiving additional delegations should not decrease their influence over the final outcome (**?**).
- **Confluence:** Guarantees that each delegating voter ends up with one clear and unambiguous delegation path. This property simplifies vote resolution and enhances transparency (**?**).

- **Copy Robustness:** Prevents strategic manipulation where a voter might mimic another's vote outside the system to gain extra influence. A copy-robust rule makes sure that duplicating a vote externally does not yield more combined power than a proper delegation (**??**).

The literature considers several delegation rules, each with distinct trade-offs:

Depth-First Delegation (DFD): Selects the path beginning with the highest-ranked delegate, even if the resulting chain is long. Although it prioritizes individual trust preferences, DFD can violate guru participation (**?**).

Breadth-First Delegation (BFD): Chooses the shortest available delegation path and uses rankings only to resolve ties. This approach usually produces direct, predictable chains and satisfies guru participation, although it might sometimes assign a vote to a lower-ranked delegate

MinSum: Balances path length and delegation quality by selecting the path with the lowest total sum of edge ranks. Being confluent, MinSum avoids both unnecessarily long chains and poorly ranked delegations (**?**).

Diffusion: Constructs delegation paths in stages by assigning votes layer by layer based on the lowest available rank at each step. This method tends to avoid poor delegations but can sometimes produce unintuitive outcomes due to its tie-breaking procedure (**?**).

Leximax: Compares paths based on their worst-ranked edge. This ensures that especially low-ranked delegations are avoided early in the path while maintaining confluence (**?**).

BordaBranching: Takes a global view of the delegation graph by selecting a branching that minimizes the total rank across all delegation edges. It satisfies both guru participation and copy robustness, though it is more computationally intensive (**?**).

In summary, ranked delegation enhances liquid democracy by reducing the risk of lost votes. The choice of delegation rule not only affects system efficiency but also influences fairness and robustness. While simpler methods such as DFD and BFD are easier to implement, advanced rules like MinSum, Leximax, and BordaBranching offer stronger guarantees and are better suited for practical deployment in platforms such as vodle.

For our implementation, MinSum will be chosen as the delegation rule because it offers a good trade-off between delegation quality, computational efficiency, and user interpretability. By selecting the path with the lowest total rank sum, MinSum prioritises well-ranked delegates while avoiding unnecessarily long or indirect delegation chains. This not only improves the quality of representation but also makes it clearer

to users why a particular delegate was chosen as the path reflects their stated preferences in a straightforward way. Additionally, MinSum is more computationally efficient than alternatives like BordaBranching, making it a practical choice for deployment within the vodle platform.

Vote Splitting

2.2 Existing Implementations of Liquid Democracy

To understand how liquid democracy can be practically integrated into vodle, it is important to examine how similar systems have been implemented in real-world contexts. This section explores two implementations, LiquidFeedback and Google Votes, that offer valuable insights into the technical, social, and usability challenges associated with applying liquid democracy at scale.

2.2.1 LiquidFeedback

LiquidFeedback is one of the earliest and most influential real-world implementations of liquid democracy. Developed as an open-source platform, it was notably adopted by the German Pirate Party in 2010 to facilitate internal policy-making through online participation (**?**). The platform allowed members to submit proposals, debate them in structured phases, and vote either directly or via transitive delegation.

In LiquidFeedback, users could choose different delegates for different topics, allowing them to assign their vote to someone they trusted on a specific issue. These choices remained in place until the user changed them, which meant that certain individuals could gradually accumulate more influence if others did not update their delegations. When multiple proposals were put forward, the system used a ranking-based voting method (such as the Schulze method) to decide which one should win. This approach compares each proposal against the others and selects the one that would win the most head-to-head matchups. Importantly, the system only accepted a proposal if it clearly beat the alternative of doing nothing, helping to avoid unnecessary or unpopular changes.

In practice, the Pirate Party's use of LiquidFeedback revealed several key dynamics relevant to this project. The platform was successful in enabling large-scale participation and crowdsourced policy formation, but it also demonstrated common risks of liquid democracy. Such as the existence of super-voters, as discussed previously.

Another practical issue was the complexity of the system. LiquidFeedback was difficult to understand for many users, especially those unfamiliar with concepts like

transitive delegation or multi-stage voting which limited its accessibility and contributed to declining engagement over time (**?**).

For a platform like vodle, the experience of LiquidFeedback highlights several important design considerations. First, user interfaces must be intuitive enough to allow voters to participate without needing deep technical knowledge. Second, the user must be know the status of their delegation at a glance - improving the understanding of the platform. Finally, ensuring that votes lead to visible and actionable outcomes is critical for sustained engagement.

2.2.2 Google Votes

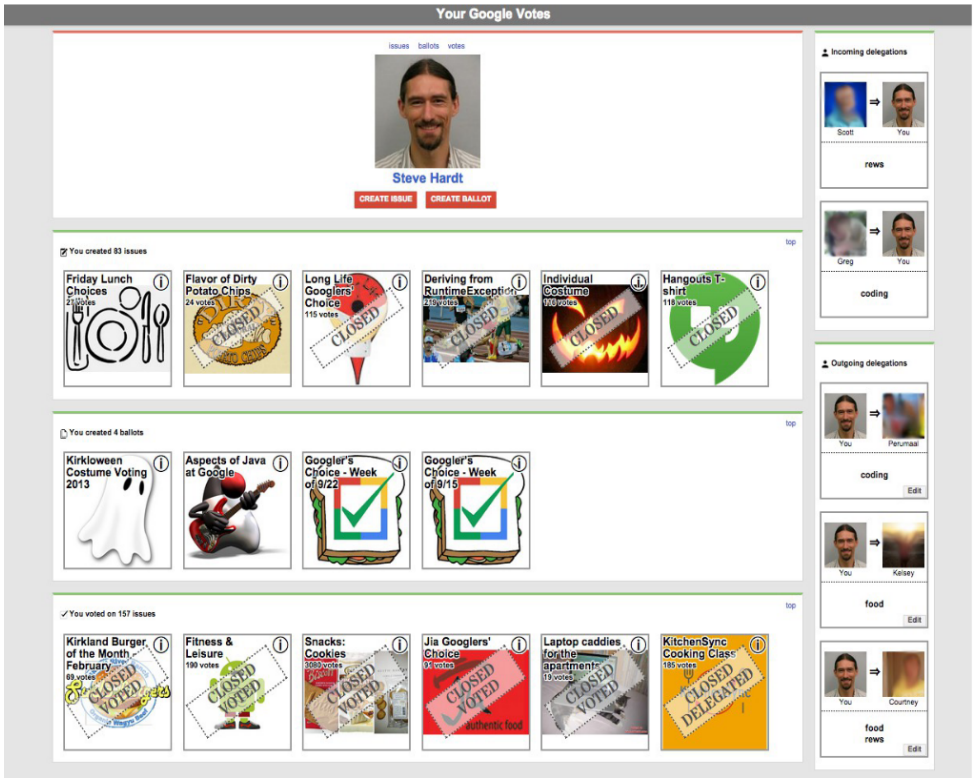

Figure 2.4: Screenshot taken from **?** showing the user interface of Google Votes.

Google Votes was an internal experiment at Google designed to explore the practical application of liquid democracy within a corporate environment. Built on top of the company's internal Google+ social network, it operated between 2012 and 2015 and allowed employees to participate in decision-making by either voting directly or delegating their vote to a colleague (**?**).

Delegations in Google Votes were category-specific, meaning that users could choose different delegates for different areas of interest, such as food, events, or technical infrastructure. These delegations were persistent but could be overridden at any

time, giving users flexibility to either rely on trusted experts or vote independently as needed. The system supported transitive delegation and allowed users to reclaim control by casting their own vote, even after delegating.

The platform placed strong emphasis on usability and transparency. Delegation features were rolled out incrementally, with additional tools such as voting power estimates and delegation advertisements helping users understand their influence. One key design principle was what the authors called the "Golden Rule of Liquid Democracy": if a user delegates their vote, they should be able to see how it is being used. To that end, users received notifications when their delegate voted, and all votes were visible to the relevant group. This encouraged accountability and gave voters confidence that their delegated votes were being used appropriately.

While Google Votes was never made publicly available, it served as a successful demonstration of liquid democracy in a structured, real-world setting. It showed that delegative voting could improve engagement and decision-making within large organisations, especially when designed with attention to user experience. For vodle, the system provides a concrete example of how features like topic-specific delegation, transparency tools, and real-time voting feedback can make liquid democracy more practical and accessible.

2.3 Agent Based Modelling

Agent-based modelling (ABM) is a computational approach used to simulate the actions and interactions of autonomous agents in order to assess their effects on a system as a whole. It is particularly suited for exploring complex, dynamic systems where behaviour emerges from local interactions between individual entities (agents) rather than being dictated by central control. ABM has been widely applied in domains such as economics, sociology, and ecology to study decentralised systems, market dynamics, and collective behaviours (**?**).

The need to explore ABM arises due to the project's goal of introducing a vote-splitting mechanism that hasn't been explored before into vodle. Traditional analysis alone may not effectively capture the dynamic interactions or unintended consequences that can emerge from this novel feature. Through ABM, it is possible to simulate realistic voting scenarios, track delegation chains, identify potential power imbalances, and anticipate challenges. These simulations can reveal performance insights and inform design decisions before implementing the mechanisms within the live platform.

Several widely used ABM frameworks exist, each with their own strengths and drawbacks relevant to this project:

- **NetLogo** (**?**) is a highly accessible and widely adopted modelling platform known for its user-friendly graphical interface and ease of learning. It offers rapid prototyping capabilities and excellent visualisation features, allowing clear communication of results. However, very complicated models are not compatible with it.

- **Repast** (**?**) provides a powerful and versatile suite of tools for building large-scale, computationally intensive simulations. It supports distributed computing, which is beneficial for extensive delegation networks with potentially thousands of agents. However, Repast has a steep learning curve, which could hinder its compatibility with this heavily time restricted project.

- **Mesa** (**?**) is an open-source framework written in Python and specifically designed for agent-based modelling. Its advantage lies in its integration with Python's ecosystem of data science libraries. Simulations built with Mesa can easily make use of tools such as NumPy and pandas for efficient data processing, and Matplotlib or Seaborn for visualising model outputs. This compatibility allows for rapid analysis and iteration, while also significantly lowering the learning curve for developers already familiar with Python. Mesa offers a practical balance between usability and computational flexibility, making it well-suited for customisable and moderately large simulations.

- **Agents.jl** (**?**) is a high-performance agent-based modelling framework written in Julia. Due to Julia's speed and efficiency, it is suitable for large-scale and computationally demanding simulations. The framework is designed to be user-friendly, with a syntax that is approachable for those familiar with scientific computing. However, the Julia ecosystem is less mature compared to Python's, which may limit the availability of additional libraries and resources.

Given the time constraints of this project, Mesa offers a practical and efficient solution. Its Python-based interface and straightforward setup allow for rapid development without the overhead of learning a new framework. This ease of use enables more time to be spent designing meaningful experiments and analysing results, rather than configuring tooling. Therefore, Mesa is the best suited framework for this project.

2.4 Vodle

Vodle is a web-based platform for participatory group decision-making. Users participate in polls that allow them to rate a set of options using sliders. When the poll ends, these ratings are aggregated and the MaxParC rating system is used to determine the final result of the poll.

2.4.1 MaxParC - TODO: summary

Understanding MaxParC is important for this project because it forms the core of how vodle interprets group preferences. Since this work involves modifying vodle's voting behaviour through the integration of liquid democracy, it is essential to understand how MaxParC processes input ratings. In particular, understanding how changes in individual ratings influence the final outcome of a poll helps to frame the implications of delegating or reweighting votes.

Maximum Partial Consensus (MaxParC), the rating system used by vodle, was introduced by **?**. It is a decision-making method designed to address the limitations of traditional voting systems, in particular the potential for majority rule to suppress minority viewpoints. The primary objective of MaxParC is to achieve a balance between fairness, consensus, and efficiency in group decision-making.

Each voter rates an option from 0 to 100 (x), representing their willingness to approve that option if and only if $< x\%$ of users do not approve that option. Therefore, a rating of 0 means "do no approve no matter what" and a rating of 100 means "approve no matter what" or "always approve". This can be visualised in the illustration below.

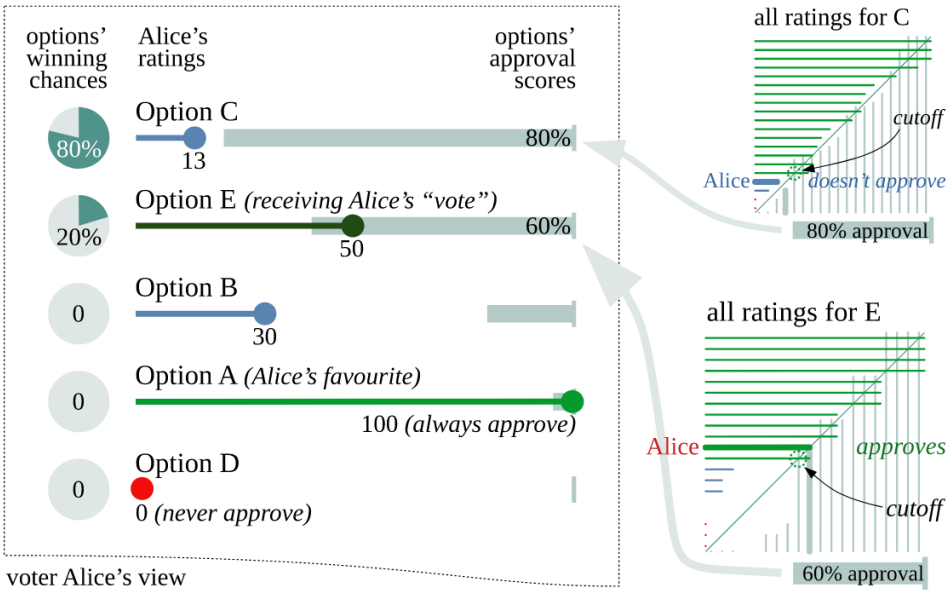

Figure 2.5: Graphic from **?** representing MaxParC from the view of some voter Alice. Each rating value represents a conditional commitment by Alice to approve the respective option. Approval scores, number of voters that approve the option, are represented by light bars coming in from the right, options are sorted by descending approval score. Alice is counted as approving an option if her rating needle overlaps with the approval score bar.

To summarise, any delegation or vote splitting mechanism must be designed so that it only adds to an option's overall rating and never reduces it. This ensures that

delegated input remains consistent with MaxParC's conditional approval model and preserves the original intent of the voter.

2.4.2 Technologies Used

Understanding vodle's technology stack is crucial to successfully integrate liquid democracy features into the existing platform. Since the project involves adding complex delegation and voting logic, it's important to appreciate the constraints and benefits of the technologies currently used in vodle, as they directly influence the design and implementation choices.

Angular

Vodle is built with Angular (**?**), a TypeScript-based frontend framework created by Google. Angular's modularity and structured component system provides a strong foundation for incremental development, essential when introducing new features such as ranked delegation that build upon existing components. Its clear separation of concerns helps maintain readable and maintainable code, simplifying debugging and future enhancements. This is particularly beneficial as the delegation logic is expected to grow in complexity later in the project.

Ionic Framework

The Ionic (**?**) framework complements Angular by enabling the creation of responsive, mobile-compatible applications from a single codebase. Given vodle's goal of broad user participation, Ionic ensures that the newly implemented liquid democracy functionalities remain consistent and accessible across both desktop and mobile devices. This cross-platform compatibility encourages greater user engagement and facilitates testing and feedback on various device types, critical for verifying usability and user interaction with delegation features.

CouchDB

CouchDB (**?**) is used as vodle's primary data store and communicates directly with the client through HTTP requests, eliminating the need for a dedicated backend server. This architecture places significant computational responsibilities on the client-side Angular application, including the handling of delegation chains, cycle detection, and the computation of final vote outcomes. Furthermore, since CouchDB stores

data exclusively as JSON-formatted strings, complex delegation structures and voting relationships must be carefully serialised and de-serialised on the client side.

The lack of server-side computation means the delegation algorithms must be designed with client-side efficiency in mind, ensuring performance remains acceptable even as delegation complexity increases. Thus, the choice of algorithms for liquid democracy features, such as those managing delegation paths and vote resolution, is directly influenced by CouchDB's architectural constraints.

These technological considerations shape the practical implementation of liquid democracy within vodle, highlighting the need for efficient client-side processing, careful data management, and cross-platform consistency.

2.4.3 Partially Implemented Delegation in Vodle

The current version of vodle includes a partially implemented delegation feature that allows users to delegate their vote to another participant in a given poll. This implementation provides the basic structure required for one-step delegation but lacks robustness in handling more complex delegation scenarios.

Delegation Workflow

1. **Creating a Delegation Link**

   User A can initiate a delegation by inviting another participant (User B) to act as their delegate. The system then generates a unique delegation link tied to the relevant poll and user A. This link can be copied manually, sent via email, or shared using a device-specific sharing interface.

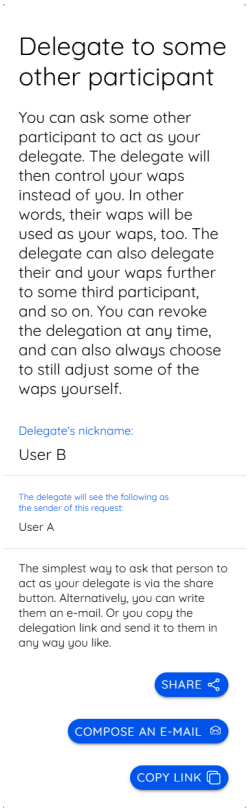

Figure 2.6: Screenshot of popup window for creating a delegation link.

Insert Code Snippet: Function for creating a delegation link and storing it in the CouchDB database.

```
function createDelegationLink(pollId: string, userId: string): string {
    const delegationLink = generateUniqueLink(pollId, userId);
    const delegationData = {
        pollId: pollId,
        userId: userId,
        status: "pending",
        link: delegationLink
    };
    storeInCouchDB(delegationData);
    return delegationLink;
}
```

2. **Accepting the Delegation**

   User B must manually accept the delegation. When the link is opened, the system prompts them with an acceptance dialog. Once accepted, the delegation document stored in CouchDB is updated with a new status (e.g., “accepted”), confirming that User B is now the official delegate for User A in that poll.

   Insert Screenshot: Acceptance screen presented to User B.

Insert Code Snippet: Function for accepting a delegation and updating its status.

3. **Final User Interface Changes**

   Once the delegation is accepted, the user interface for both User A and User B is updated to reflect the new delegation status. User A's voting interface will indicate that their vote has been delegated, while User B's interface will show that they are now acting as a delegate for another user.

   After a successful delegation:

   - User A sees a message indicating that their vote has been delegated and can no longer interact with the voting interface for that poll.

     Insert Screenshot: User A's view after delegation.

   - User B is notified that they are voting on behalf of another user. The interface reflects this with visual indicators showing the number of users their vote is used for.

     Insert Screenshot: User B's view while acting as a delegate.

Current Limitations

Although the current system includes logic intended to prevent delegation cycles, it does not function reliably in all cases. While it can block simple cases like direct self-delegation, more complex cycles involving multiple users are not consistently detected. For instance, in a chain where User A delegates to B, B to C, and C back to A, the system may fail to recognise the cycle depending on ...

Insert Code Snippet: Example showing the existing cycle detection logic and where it may fail.

This inconsistency can result in cycles forming unnoticed, potentially leading to lost votes or undefined behaviour in the voting system. Improving this cycle detection mechanism is a key part of this project.

2.4.4 Design Philosophy

2.5 Summary

Chapter 3

Project Objectives

This chapter outlines the core goals that guide the implementation and evaluation of liquid democracy features within vodle. These objectives were derived from the background research and are designed to address the technical and theoretical challenges identified with traditional delegation systems.

To manage the project effectively, the objectives are divided into two categories: **core objectives**, which form the backbone of the implementation and are essential to meeting the project's main goals, and **extension objectives**, which build on the core features to provide additional insight or value.

Each objective is later broken down into specific functional and non-functional requirements. This structure helps to clarify expectations, guide implementation, and provide clear criteria for evaluating whether each objective has been met. Core Objectives:

1. **Implement a Core Delegation Model into Vodle:** Build upon the existing, partially implemented delegation code within the vodle platform to create a fully functional system, including resolving key challenges such as cyclic delegations.
2. **Implement Ranked Delegation into Vodle:** Add a backup delegation mechanism to vodle, allowing users to specify an ordered list of up to 3 delegates, and using the MinSum algorithm to determine the activated delegation paths.
3. **Implement a Vote Splitting Delegation Mechanism into Vodle:** Add functionality to vodle to delegate fractions of their rating to different delegates. Use the *will come back to when research written up* system to calculate final ratings.
4. **Implement the Ability to Delegate Individual Options to Different Users**: Allow users to delegate the ratings of specific options to different delegates.

Extension Objective:

1. **Simulate Delegation Mechanisms:** Perform agent-based modelling to analyse the effectiveness of various delegation systems, including those outlined in Objectives 1, 2 and 3.

3.1 Project Requirements

The project objectives represent high-level goals that must be translated into specific, actionable requirements. This process is essential for clarifying the scope of the work, ensuring comprehensive coverage of each objective, and establishing a structured foundation for both implementation and evaluation.

To support this, requirements are organised into two categories: functional (F) and non-functional (NF). Functional requirements describe the core behaviours and features the system must support, while non-functional requirements define performance, usability, and other quality-related constraints (**?**). Distinguishing between these categories helps ensure that both the system's functionality and overall user experience are properly addressed.

Each requirement is formulated to be measurable and testable. This allows for objective evaluation during development, facilitates verification against the project goals, and helps identify areas for improvement as the system evolves.

3.1.1 Implement a Core Delegation Model into Vodle

Functional Requirements

- **FR1:** The system shall correctly handle the delegation process from invitation to acceptance.
  - **FR1.1:** The system shall allow users to invite others to act as their delegate.
  - **FR1.2:** The system shall allow invited users to accept delegation requests.
  - **FR1.3:** The system shall prevent users from accepting their own delegation invitations.
  - **FR1.4:** The system shall detect and prevent the formation of delegation cycles.
- **FR2:** The system shall provide users with a clear view of their current delegations, including the ability to revoke any delegation at any time.
- **FR3:** The system shall resolve delegations transitively, such that if User A delegates to B and B delegates to C, User C is the final casting voter for A's vote.

- **FR4:** The system shall allow users to override a delegate's decision for specific poll options by submitting a direct vote.
- **FR5:** The user interface for delegation shall be intuitive and accessible, with clear instructions and minimal friction to perform delegation actions.

Non-Functional Requirements

- **NFR1:** All delegation-related data must be stored in a JSON-encoded format, ensuring compatibility with the existing vodle CouchDB database.
- **NFR2:** Any changes to the database schema must be backward compatible, ensuring that existing data is not lost or corrupted during the upgrade process.
- **NFR3:** Any additional data stored in the database must be encrypted, using the same encryption method as the existing data, to ensure user privacy and security.
- **NFR3:** The system shall preserve user privacy by ensuring that individual voting preferences and delegation choices are not visible to other users. The only information visible to a delegated user shall be the final vote cast on their behalf.

3.1.2 Implement Ranked Delegation into Vodle

Functional Requirements

- **FR1:** The system shall allow users to specify a ranked list of up to 3 delegates for each poll, with the ranking applying to all options within that poll.
- **FR2:** The system shall apply the MinSum delegation rule to resolve each voter's delegation path based on their ranked list of delegates.
- **FR3:** The system shall allow users to override the ranked delegation by submitting a direct vote for specific poll options.
- **FR4:** The system shall provide users with a clear view of their ranked delegation choices, including the ability to alter the rankings or revoke them at any time.

Non-Functional Requirements

- **NFR1:** The system shall ensure that the ranked delegation process does not introduce significant latency in the voting process, maintaining a response time of less than 2 seconds for delegation-related actions.

- **NFR2:** The user interface for ranked delegation shall be intuitive and accessible, with clear instructions and minimal friction to perform delegation actions.
- **NFR3:** The system shall ensure that any data related to ranked delegation is stored in a JSON-encoded format, ensuring compatibility with the existing vodle CouchDB database.

3.1.3 Implement a Vote Splitting Delegation Mechanism into Vodle

Functional Requirements

- **FR1:** The system shall allow users to delegate their vote to multiple delegates simultaneously for a single poll.
- **FR2:** The system shall allow users to assign a weight to each delegate such that the total weight does not exceed 0.99.
- **FR3:** The system shall use the algorithm described in **??** to calculate the final rating for each option based on the weights assigned to each delegate.
- **FR5:** The system shall provide users with a visual interface to edit the weights assigned to each delegate, with either sliders or numeric inputs for easy adjustment.

Non-Functional Requirements

- **NFR1:** The system shall ensure that vote-splitting calculations are performed entirely on the client side to comply with vodle's CouchDB architecture.
- **NFR2:** Delegation weights and related data must be serialised as JSON strings to ensure compatibility with the CouchDB backend.
- **NFR3:** The user interface for vote splitting shall be intuitive and allow users to adjust weights easily, using sliders or numeric inputs.

3.1.4 Implement the Ability to Delegate Individual Options to Different Users

Functional Requirements

- **FR1:** The system shall allow users to assign different delegates for each individual option or subset of options within a poll.

- **FR2:** The system shall ensure that each delegated option is resolved independently, using the appropriate delegate's vote for that option.
- **FR3:** The system shall allow users to override a delegate's vote for a specific option by submitting their own rating.
- **FR4:** The system shall provide a user interface for viewing or revoking each individual delegation.

Non-Functional Requirements

- **NFR1:** The delegation interface must be intuitive and clearly indicate which delegate is assigned to each option, ensuring ease of use even for users unfamiliar with delegation models.
- **NFR2:** The delegation data must be serialised in a format compatible with CouchDB (e.g., JSON-encoded) to maintain compatibility with vodle's storage system.

3.1.5 Simulate Delegation Mechanisms

Functional Requirements

- **FR1:** The simulation system shall model individual agents representing voters, each capable of voting, abstaining, or delegating their vote according to a selected delegation rule.
- **FR2:** The system shall support multiple delegation mechanisms, including standard transitive delegation, ranked delegation (with the MinSum delegation rule), and vote splitting.
- **FR3:** The system shall allow configuration of simulation parameters such as number of agents, delegation probabilities and abstention rates.
- **FR4:** The system shall track and record key metrics such as vote concentration, number of super-voters, average chain length, vote loss due to abstentions or cycles, and decision quality.
- **FR5:** The system shall output simulation results in a structured format (e.g. CSV or JSON) for further analysis.

Non-Functional Requirements

- **NFR1:** The simulation framework must be lightweight and easy to extend, enabling rapid experimentation with new delegation rules or metrics.
- **NFR2:** The system shall be developed using Mesa to take advantage of existing data science libraries such as NumPy, Pandas, and Matplotlib for analysis and visualisation.
- **NFR4:** The simulation design shall support reproducibility by enabling fixed random seeds and storing configuration settings alongside output data.

Chapter 4

Design and Implementation (can split into separate chapters)

Chapter 5

Evaluation

Chapter 6

Project Management

This chapter outlines the project's management approach, including the development methodology, planning, and reflections on the process. It also considers legal and ethical issues and assesses key risks associated with the project.

6.1 Methodology

The project followed an agile methodology, selected for its flexibility and its emphasis on iterative development and regular customer feedback. The work involved building a sequence of interdependent features into vodle; starting with a basic delegation mechanism and expanding to include ranked delegation, vote splitting, and per-option delegation. Since each feature built on the last, an iterative approach was necessary to ensure compatibility and to allow design decisions to adapt over time.

Agile was particularly well suited to this project due to the presence of an active "customer" figure: Jobst Heitzig, the co-supervisor and original creator of vodle. His role went beyond academic supervision - he provided clarified system expectations and helped shape design decisions based on the real-world use case. Fortnightly meetings were held with both Jobst and the academic supervisor, Markus Brill, to review progress and incorporate their feedback into the next development cycle. This tight feedback loop is a core principle of agile, allowing the project to stay aligned with user needs and system goals.

Alternative models such as Waterfall were considered but ultimately rejected due to their inflexibility. Waterfall could have provided clear documentation at each phase and well-defined milestones, which initially seemed beneficial for implementing interconnected voting delegation features. However, Waterfall requires defining the full scope of the project upfront and offers limited room for revision - something that would have been impractical due to the project's shorter timeframe and the evolving

nature of requirements as features were tested. Agile allowed the system to evolve in parallel with the design and testing of new ideas.

Elements of Scrum were also considered, as it is one of the most widely adopted agile frameworks, used by 63% of teams use Scrum (**?**). The sprint-based development cycles, clearly defined roles, and structured ceremonies like sprint planning and reviews were attractive features that could have facilitated focused implementation and effective communication. However, daily stand-up meetings and fixed-length sprints were not feasible for this project, as all parties involved (myself and the two supervisors) had other commitments. The small team size didn't warrant all Scrum ceremonies, and the academic nature of the project called for more flexible review cycles. Instead, progress was reviewed every two weeks, ensuring feedback could still be gathered and acted upon without adding unnecessary scheduling pressure.

Each development cycle produced a working, testable feature that could be evaluated and integrated into the overall system. This approach reduced the risk of late-stage errors and helped maintain steady progress throughout the project. Agile's iterative and feedback-driven structure was a natural fit for the technical and collaborative demands of this work.

6.2 Plan

The project plan was organised into objectives (see section **??**) that built on one another in sequence:

- **Core Objective 1:** Implement a Core Delegation Model into Vodle.
- **Core Objective 2:** Implement Ranked Delegation into Vodle.
- **Core Objective 3:** Implement a Vote Splitting Delegation Mechanism into Vodle.
- **Core Objective 4:** Implement the Ability to Delegate Individual Options to Different Users.
- **Extension Objective 1:** Simulate Delegation Mechanisms.

This objective-led structure was well-suited to the agile approach, allowing each milestone to be treated as an iteration with a deliverable at the end. A Gantt chart (see below) was created to visualise the project timeline and to track dependencies and progress.

TODO: Gantt Chart

6.3 Risk Assessment - TODO

Risk	**Likelihood**	**Mitigation Strategy**
Breaking the live vodle site during development	Medium	Use Git branching to isolate development from production environments. Conduct local testing before deployment.
Feature complexity exceeds estimates	High	Prioritise core objectives and maintain flexibility in scope.
Lack of engagement from supervisors or stakeholders	Low	Maintain regular communication through scheduled meetings.
Data loss or corruption	Low	Use Git for version control and take regular local backups.

Table 6.1: Key risks identified and mitigation strategies

6.4 Legal and Ethical Considerations

As vodle may eventually be used to gather votes on sensitive topics, care was taken to ensure privacy and fairness. Delegation chains are resolved internally and never publicly exposed, preserving the confidentiality of voter relationships. No personal data was collected or processed for the purposes of this project, so no changes to the platform's terms of service were required.

6.5 Risk Management Reflection - TODO

6.6 Overall/Self Reflection - TODO

Chapter 7

Conclusions