Context Closure: Concise Overview

December 15, 2025

Motivation

- Documentation depends on code, tickets, and discussions; signals are unbounded.

Motivation

- Documentation depends on code, tickets, and discussions; signals are unbounded.
- LLM context windows (and human attention) are bounded.

Motivation

- Documentation depends on code, tickets, and discussions; signals are unbounded.
- LLM context windows (and human attention) are bounded.
- Scalability: we must pick the most valuable, self-contained context slice, not dump everything.

Motivation

- Documentation depends on code, tickets, and discussions; signals are unbounded.
- LLM context windows (and human attention) are bounded.
- Scalability: we must pick the most valuable, self-contained context slice, not dump everything.
- Goal: organize signals to maximize documentation value within hard context limits.

Scalability $\Rightarrow$ Graph + Optimal Closure

- Chunk the data (messages, issues, PRs, doc gaps) into nodes; remove duplication via dependencies (edges: $u \rightarrow v$ means u needs v).

Scalability $\Rightarrow$ Graph + Optimal Closure

- Chunk the data (messages, issues, PRs, doc gaps) into nodes; remove duplication via dependencies (edges: $u \to v$ means u needs v).
- Assign utility weight per chunk; higher = more value for the next doc update.

Scalability $\Rightarrow$ Graph + Optimal Closure

- Chunk the data (messages, issues, PRs, doc gaps) into nodes; remove duplication via dependencies (edges: $u \to v$ means u needs v).
- Assign utility weight per chunk; higher = more value for the next doc update.
- Select k chunks that form a closure (no outgoing edges), maximizing total weight.

Scalability $\Rightarrow$ Graph + Optimal Closure

- Chunk the data (messages, issues, PRs, doc gaps) into nodes; remove duplication via dependencies (edges: $u \to v$ means u needs v).
- Assign utility weight per chunk; higher = more value for the next doc update.
- Select k chunks that form a closure (no outgoing edges), maximizing total weight.
- This construction **scales** because it enforces bounded chunks, avoids duplication, and optimizes for value under hard context limits.

Scalability $\Rightarrow$ Graph + Optimal Closure

- Chunk the data (messages, issues, PRs, doc gaps) into nodes; remove duplication via dependencies (edges: $u \to v$ means u needs v).
- Assign utility weight per chunk; higher = more value for the next doc update.
- Select k chunks that form a closure (no outgoing edges), maximizing total weight.
- This construction **scales** because it enforces bounded chunks, avoids duplication, and optimizes for value under hard context limits.
- This is exactly the **maximum-weight closure** problem.

Optimal Closure Primer

- Given a directed graph with weights, find a closed subset (no outgoing edges) with maximum total weight.

Optimal Closure Primer

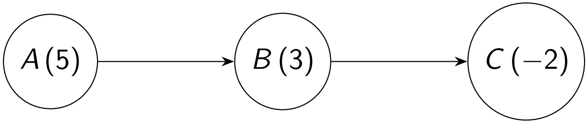
- Given a directed graph with weights, find a closed subset (no outgoing edges) with maximum total weight.
- Solvable via min-cut reduction (polynomial time).

Optimal Closure Primer

- Given a directed graph with weights, find a closed subset (no outgoing edges) with maximum total weight.
- Solvable via min-cut reduction (polynomial time).
- Our use: pick the best self-contained context under size/budget constraints.

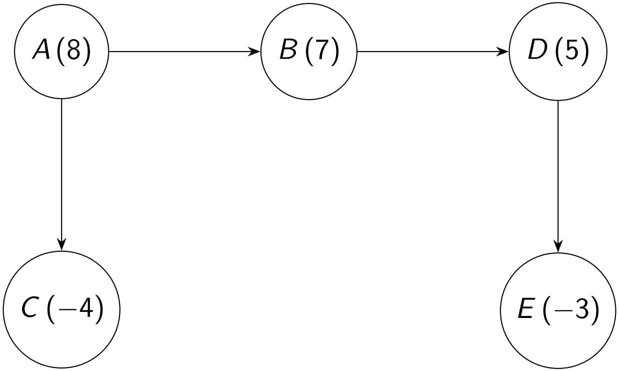
Example 1 (tiny)

$$\begin{tikzcd} A\,(5) \arrow[r] & B\,(3) \arrow[r] & C\,(-2) \end{tikzcd}$$



Optimal closure: $\{A, B, C\}$ (weight 6). This is not hard to inspect.

Example 2 (medium)



Optimal closure: $\{A, B, D\}$ (weight 20). This is not hard to inspect, but certainly more complicated than the last one.

Example 3 (larger) ... this problem can be difficult.

$$\begin{tikzcd}
A\,(6) \arrow[r] \arrow[d] & B\,(4) \arrow[r] \arrow[d] & D\,(9) \arrow[r] & G\,(2) \arrow[r] \arrow[dd] & H\,(-6) \\
C\,(-5) \arrow[rr] & E\,(7) \arrow[r] \arrow[dr] & F\,(-3) \arrow[d] & & \\
 & & I\,(5) \arrow[r] & J\,(4) \arrow[dl, no head] & \\
 & & {} & &
\end{tikzcd}$$

Generalization

- This construction actually solves a more general problem than preparing LLM context windows.
- k need not be an LLM context limit; it can be any resource budget (time, people, memory).

Generalization

- This construction actually solves a more general problem than preparing LLM context windows.
- k need not be an LLM context limit; it can be any resource budget (time, people, memory).
- Weights can encode any utility (task priority, revenue, risk).

Generalization

- This construction actually solves a more general problem than preparing LLM context windows.
- k need not be an LLM context limit; it can be any resource budget (time, people, memory).
- Weights can encode any utility (task priority, revenue, risk).
- Closure selection still yields the highest-value, self-contained subset for the chosen budget.

Generalization

- This construction actually solves a more general problem than preparing LLM context windows.
- k need not be an LLM context limit; it can be any resource budget (time, people, memory).
- Weights can encode any utility (task priority, revenue, risk).
- Closure selection still yields the highest-value, self-contained subset for the chosen budget.
- LLM-driven docs are one special case; the method applies to broader task selection.

Drawbacks & Mitigations

- Dependency detection may be incomplete or noisy $\rightarrow$ improve link extraction, allow human corrections.

Drawbacks & Mitigations

- Dependency detection may be incomplete or noisy $\rightarrow$ improve link extraction, allow human corrections.
- Super heavy reliance on the Weightings, which may be biased or stale. $\rightarrow$ add recency decay, source trust, feedback loops.

Drawbacks & Mitigations

- Dependency detection may be incomplete or noisy $\rightarrow$ improve link extraction, allow human corrections.
- Super heavy reliance on the Weightings, which may be biased or stale. $\rightarrow$ add recency decay, source trust, feedback loops.
- Graph churn and scale $\rightarrow$ incremental updates, caching, and persisted graphs.

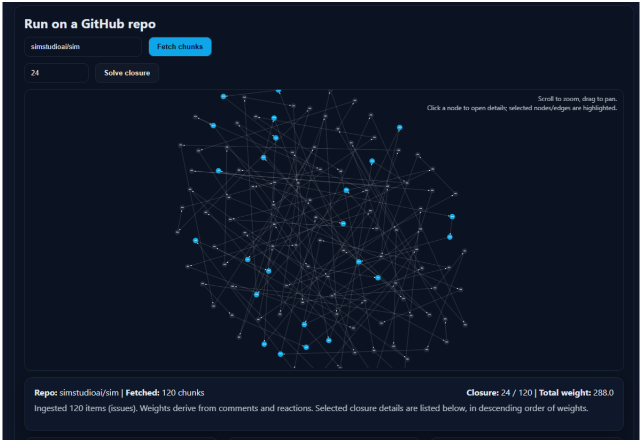
Drawbacks & Mitigations

- Dependency detection may be incomplete or noisy $\rightarrow$ improve link extraction, allow human corrections.
- Super heavy reliance on the Weightings, which may be biased or stale. $\rightarrow$ add recency decay, source trust, feedback loops.
- Graph churn and scale $\rightarrow$ incremental updates, caching, and persisted graphs.
- Empty or sparse edges $\rightarrow$ fallback visualization only (not for optimization), keep optimization on real edges.

Drawbacks & Mitigations

- Dependency detection may be incomplete or noisy $\rightarrow$ improve link extraction, allow human corrections.
- Super heavy reliance on the Weightings, which may be biased or stale. $\rightarrow$ add recency decay, source trust, feedback loops.
- Graph churn and scale $\rightarrow$ incremental updates, caching, and persisted graphs.
- Empty or sparse edges $\rightarrow$ fallback visualization only (not for optimization), keep optimization on real edges.

System Screenshots I


Run on a GitHub repo
simstudioai/sim
Fetch chunks
24
Solve closure
Scroll to zoom, drag to pan.
Click a node to open details; selected nodes/edges are highlighted.

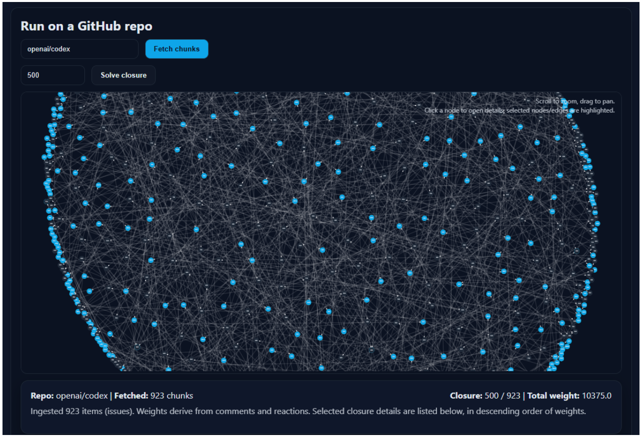
Repo: simstudioai/sim | **Fetched:** 120 chunks **Closure:** 24 / 120 | **Total weight:** 288.0

Ingested 120 items (issues). Weights derive from comments and reactions. Selected closure details are listed below, in descending order of weights.

Figure: Example: simstudioai/sim

This is a moderately-sized example of a size 24 closure against 120 total nodes.

System Screenshots II


Run on a GitHub repo
openai/codex
Fetch chunks
500
Solve closure
Scroll to zoom, drag to pan.
Click a node to open details; selected nodes/edges are highlighted.

Repo: openai/codex | **Fetched:** 923 chunks **Closure:** 500 / 923 | **Total weight:** 10375.0

Ingested 923 items (issues). Weights derive from comments and reactions. Selected closure details are listed below, in descending order of weights.

Figure: Example: openai/codex

This is a huge example of a size 500 closure against 923 total nodes.

Signal Processing & Storage

- Ingest: GitHub issues/PRs (all open), simple weights (comments + reactions).

Signal Processing & Storage

- Ingest: GitHub issues/PRs (all open), simple weights (comments + reactions).
- Transform: chunk records with metadata (source, component, tags); edges from dependencies when available.

Signal Processing & Storage

- Ingest: GitHub issues/PRs (all open), simple weights (comments + reactions).
- Transform: chunk records with metadata (source, component, tags); edges from dependencies when available.
- Store: in-memory per request (current prototype); deterministic layout for graph rendering.

Signal Processing & Storage

- Ingest: GitHub issues/PRs (all open), simple weights (comments + reactions).
- Transform: chunk records with metadata (source, component, tags); edges from dependencies when available.
- Store: in-memory per request (current prototype); deterministic layout for graph rendering.
- Serve: API returns graph; closure solver returns selected nodes; UI renders graph and selected chunk details.