The Mesh Digital Agora Deliberation System Architecture

Abstract. Mesh is a structured discourse system that represents conversation as a graph of typed entities. Claims exist as stable objects with unique identifiers. Evidence binds to assertions through explicit citation links. Challenges target specific premises, inferences, or conclusions rather than messages as a whole. The system supports progressive formalization: participants begin with informal discussion and introduce structure incrementally as complexity warrants. Multiple surfaces—publishing, curation, discussion, and deliberation—share a common foundation, so work performed in one context becomes available in others. Arguments constructed in one deliberation can migrate into related deliberations with provenance preserved. The network of deliberations forms a higher-level graph where cross-references, overlapping claims, shared sources, and imported arguments create explicit relationships. As participants contribute to this shared structure, the system accumulates institutional memory: conclusions traceable to their grounds and arguments available for reuse across contexts.

1. Introduction

Mesh provides infrastructure for collective inquiry. When groups need to coordinate around complex questions—evaluating policy options, analyzing technical tradeoffs, reaching shared conclusions—they require more than message exchange. They require a way to see the structure of a discussion: what has been claimed, what supports each claim, where disagreements lie, and what remains unresolved.

Mesh addresses this need by representing discourse as a graph rather than a sequence of messages. Every assertion promoted into the system becomes an addressable object. Every piece of evidence attaches explicitly to the claims it supports. Every challenge specifies its target: a premise, an inference step, or a conclusion. The result is a deliberation that participants can inspect, extend, and reference—a record that persists beyond the conversation and compounds over time.

The system is designed around progressive formalization. A discussion can remain informal indefinitely—real-time chat for coordination, threaded posts for durable exchange. When a thread becomes consequential, participants can introduce structure incrementally: promoting assertions into claims, attaching evidence, constructing arguments, and composing outputs. The platform accommodates both casual check-ins and multi-year institutional deliberations within the same architecture.

Mesh exposes multiple surfaces that share a common foundation. Publishing provides longform writing with annotation and deliberation hosting. Curation provides source collection with deduplication and citation workflows. Discussion provides informal exchange with upgrade paths to deliberation. Deliberation provides structured argument construction with artifact production. Discovery provides a feed of activity and a network of relationships across deliberations. These surfaces are entry points into the same underlying system, so claims created in one context can be cited, supported, challenged, and reused in others.

Architecture at a Glance

The system organizes reasoning into three layers:

Layer	Purpose	Key Artifacts
Claims & Evidence	Transform informal ideas into canonical, evidence-linked assertions	Propositions, Claims, ClaimEdges, Evidence
Arguments & Dialogue	Structure reasoning with premises, conclusions, schemes, and tracked moves	Arguments, ArgumentChains, DialogueMoves, Commitments
Outputs & Artifacts	Compose reasoning into publishable, citable documents	Thesis, TheoryWorks, KbPages, DebateSheets

All structures conform to the **Argument Interchange Format (AIF)** ontology, enabling interoperability with academic argumentation tools and ensuring that reasoning graphs can be exported, analyzed, and verified independently.

For Developers

This document serves as both conceptual orientation and implementation reference. The central UI orchestration component is `DeepDivePanelV2` (located at `components/deepdive/DeepDivePanelV2.tsx`), which connects the subsystems described herein through a tabbed interface with floating sheets for deep exploration. Understanding this component provides a practical entry point into the codebase.

Table of Contents

---

1. Global System Design

This section presents the big-picture architecture—the conceptual framework that explains *why* the system is structured the way it is, before diving into implementation details. This section reflects the **current implemented state** of the platform as of December 2024.

1.1 The Problem We Solve

Why Good Decisions Are Hard to Make Together

Every organization—whether a company, a government agency, a nonprofit, or a community group—faces the same fundamental challenge: **how do you reason well together?**

Good collective decisions require several things that are surprisingly difficult to achieve:

1. **You need to see the reasoning, not just the conclusion.** When someone proposes "we should do X," you need to understand *why*—what evidence supports it, what assumptions underlie it, what alternatives were considered. Without this, you can't evaluate the decision or improve it.

2. **You need to navigate complexity.** Real questions generate dozens of claims, each supported or challenged by others, referencing different sources. Linear documents and threaded comments can't represent this structure. People either oversimplify or get lost.

3. **You need to know who said what and why.** Ideas shouldn't float anonymously. Every claim should have an author, a source, and a context. When claims conflict, there should be a clear way to trace the disagreement.

4. **You need arguments to get better, not just louder.** When someone finds a flaw in reasoning, there should be a path to address it—to strengthen the premise, acknowledge the limitation, or revise the conclusion. Most platforms reward winning arguments, not refining them.

5. **You need to build on prior work.** When a team spends months analyzing a question, that analysis shouldn't disappear when the project ends. Future teams facing similar questions should be able to find, reference, and extend that work.

These aren't exotic requirements. They're what thoughtful reasoning looks like. But current digital tools make all five nearly impossible.

What's Broken Today

The platforms where groups discuss important questions—email threads, Slack channels, Google Docs, comment sections—were not designed for structured reasoning. They're optimized for conversation, not deliberation.

Problem	What Happens	What It Costs
No shared structure	Ideas exist as prose, not as claims with explicit relationships	Disagreements become rhetorical battles rather than structured analysis
Evidence is disconnected	Sources live in attachments, footnotes, or separate documents	No way to trace a claim back to its grounds, or see what depends on disputed evidence
No institutional memory	Reasoning scatters across threads, docs, meetings, and email	Every discussion starts from scratch; past conclusions can't be cited or audited
No path to resolution	Comments pile up but don't converge toward conclusions	Decisions get made elsewhere (or not at all), without clear justification

These aren't just inconveniences. They erode the ability of institutions to justify their decisions, of teams to learn from past analysis, and of communities to resolve disagreements constructively.

What We Build Instead

The solution isn't "better comments." It's infrastructure for reasoning—a method and data model that treats arguments as structured objects rather than unstructured text.

Agora provides:

- **Canonical claims**: Discrete assertions with stable identifiers that persist across contexts. When you reference "Claim #247," everyone is talking about the same thing.
- **Typed relationships**: Explicit structure showing how claims relate. Does A *support* B, *rebut* it, *undercut* the inference, or *undermine* a premise? The system distinguishes these.
- **Scheme-based reasoning**: Arguments built using recognized patterns (e.g., argument from expert opinion, argument from analogy) with automatically surfaced questions that challenge each pattern.
- **Dialogue tracking**: Every assertion, challenge, and concession recorded as a move in an ongoing dialogue, with full provenance (who, when, in response to what).
- **Composable outputs**: Deliberations produce documents, knowledge base pages, and exportable graphs that persist as institutional memory—citable, auditable, and extensible

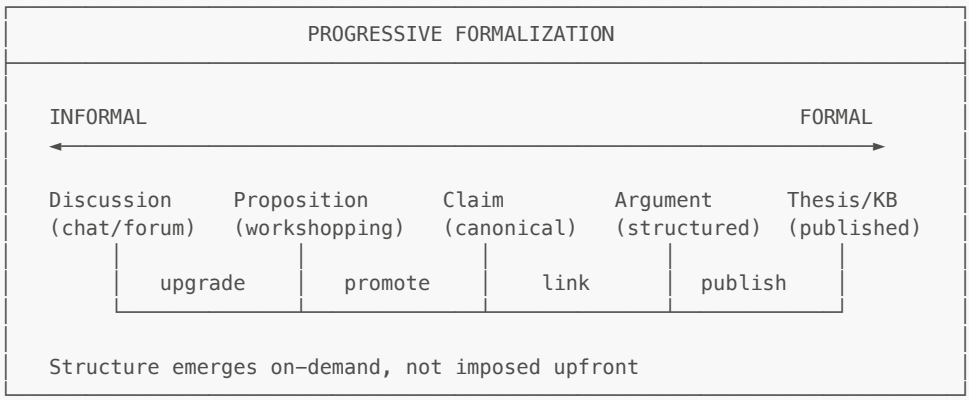
1.2 Core Design Philosophy

The architecture embodies four key principles that distinguish Agora from both traditional forums and document-centric collaboration tools.

Progressive Formalization

Not all conversations require formal structure. A team checking in on project status doesn't need argumentation schemes. A policy working group evaluating regulatory options does.

The system applies **progressive disclosure**: structure activates only when complexity warrants it. Users can stay in lightweight discussion mode indefinitely, or upgrade to full deliberation when formalization becomes valuable. The same platform serves casual check-ins and multi-year institutional deliberations.



Currently Implemented Transitions:

- `Discussion → Deliberation`: Via `DeliberateButton` component, creates linked deliberation space
- `Proposition → Claim`: Via `PromoteToClaimButton`, elevates validated propositions to canonical claims
- `Claim → Argument`: Via `AIFArgumentWithSchemeComposer`, creates structured arguments with scheme
- `Argument → Thesis`: Via `ThesisComposer`, composes claims/arguments into legal-style documents

AIF-Centric Architecture

Academic argumentation research has produced a standard for representing argument structures: the **Argument Interchange Format (AIF)**. Rather than inventing proprietary representations, Agora implements AIF as its canonical data model. This provides:

- **Interoperability**: Export to tools used in argumentation research and formal verification
- **Theoretical grounding**: Decades of research on argument semantics, attack types, and extension computation
- **Explicit structure**: Clear distinction between content (I-nodes), inference (RA-nodes), conflict (CA-nodes), and preference (PA-nodes)

AIF Node Type	Description	Implementation

AIF Node Type	Description	Implementation
I-node (Information)	Propositions/claims containing content	`Claim`, `AifNode` (nodeKind='I')
RA-node (Rule of Application)	Inference steps applying schemes	`Argument`, `AifNode` (nodeKind='RA')
CA-node (Conflict Application)	Attack relations (rebut/undercut/undermine)	`ArgumentEdge`, `AifNode` (nodeKind='CA')
PA-node (Preference Application)	Priority/ordering relations	`AifNode` (nodeKind='PA')
DM-node (Dialogue Move)	Locutions in dialogue (WHY, GROUNDS, CONCEDE)	`DialogueMove`, `AifNode` (nodeKind='DM')

Dialogue-First Reasoning

Arguments don't exist in isolation—they emerge through **structured dialogue**. Someone asserts a claim. Another participant challenges it ("WHY do you believe that?"). The original author provides grounds. A third party undercuts the inference. Each of these is a **dialogue move** with explicit semantics.

This dialogue-first approach means every structure in the system has provenance: who created it, when, in response to what. Arguments are not free-floating logical objects but moves in an ongoing conversation.

```
DialogueMove (locution) → creates → AifNode (content)
                       → records → who said what, when, in reply to what
```

Implemented Move Types: ASSERT, WHY, GROUNDS, CONCEDE, RETRACT, CLOSE

Confidence & Uncertainty

Real-world reasoning involves uncertainty. Premises may be likely rather than certain. Evidence may be partial. The system treats confidence as first-class, tracking it at multiple levels:

Scope	Implementation	UI
Per-argument	`Argument.confidence` (0.0-1.0)	Confidence sliders in composers
Aggregation mode	`DeliberationState.confMode`	Product / Min toggle
Dempster-Shafer intervals	`Deliberation.dsMode`	DS Mode toggle in header
Temporal decay	`Argument.lastUpdatedAt`	Stale badges on old arguments

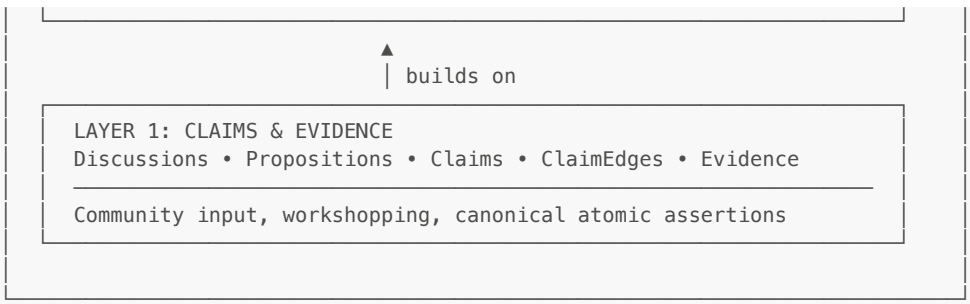
This allows the system to distinguish between strongly-supported conclusions and tentative hypotheses, and to surface when previously confident conclusions have become stale due to lack of recent validation.

1.3 The Three Conceptual Layers

The platform organizes reasoning into three distinct but connected layers. Each layer builds on the one below, adding structure and formalization as needed. A deliberation might use only Layer 1 (informal discussion with workshopping). A policy analysis might span all three, culminating in published thesis documents that cite structured arguments grounded in canonical claims.

This layered architecture reflects how reasoning actually works: ideas start informal, get refined through dialogue, connect into argument structures, and eventually crystallize into outputs that persist beyond the conversation.

```
┌─────────────────────────────────────────────────────────────────────┐
│                      THREE CONCEPTUAL LAYERS                        │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│   ┌─────────────────────────────────────────────────────────────┐   │
│   │  LAYER 3: OUTPUTS & ARTIFACTS                               │   │
│   │  Thesis • TheoryWorks • DebateSheets • KbPages • Briefs     │   │
│   │  ─────────────────────────────────────────────────────────  │   │
│   │  Structured documents, visual maps, publishable knowledge   │   │
│   └─────────────────────────────────────────────────────────────┘   │
│                              ▲                                      │
│                              │ composes                             │
│   ┌─────────────────────────────────────────────────────────────┐   │
│   │  LAYER 2: ARGUMENTS & DIALOGUE                              │   │
│   │  Arguments • ArgumentChains • Schemes • DialogueMoves • Commitments │
│   │  ─────────────────────────────────────────────────────────  │   │
│   │  Scheme-based reasoning, attack/defense, formal dialogue protocol │
```



Layer 1: Claims & Evidence (Currently Implemented)

Purpose: Transform informal ideas into canonical, evidence-linked assertions.

The foundation of structured reasoning is the **claim**—a discrete assertion that can be referenced, linked, attacked, and defended. But claims don't appear fully-formed. They emerge from informal discussion, get refined through community workshopping, and only graduate to canonical status when they've proven their value.

This layer provides the path from "someone said something in a meeting" to "we have a stable assertion we can build arguments around."

Component	Model	Key Features
Discussions	`Discussion`	Forum-style threads, chat mode, upgradable to deliberation
Propositions	`Proposition`	Workshopping with votes (`PropositionVote`), endorsements, replies
Claims	`Claim`	Canonical assertions with MOID identifiers, negation relations
ClaimEdges	`ClaimEdge`	Typed relations: SUPPORTS, REBUTS, UNDERCUTS, UNDERMINES
Evidence	`ClaimEvidence`	Citations with confidence, source URLs, CSL metadata
Contraries	`ClaimContrary`	ASPIC+ explicit contrary relations between claims

Key Components:

- `PropositionComposer` / `PropositionComposerPro`: Create propositions with evidence
- `PromoteToClaimButton`: Elevate validated propositions to canonical claims
- `ClaimMiniMap` / `CegMiniMap`: Visual navigation of claim relationships
- `ClaimDetailPanel`: Deep-dive into individual claims with CQ status

Layer 2: Arguments & Dialogue (Currently Implemented)

Purpose: Structure reasoning with explicit premises, conclusions, and inferential steps.

Claims alone don't make arguments. An argument connects claims through inference: "Given premises A and B, therefore conclusion C." The strength of this inference can be challenged. The premises can be disputed. The inference pattern itself can be questioned.

Layer 2 provides the machinery for structured argumentation. Arguments reference **schemes**—recognized patterns of reasoning (argument from expert opinion, argument from analogy, argument from cause to effect) with **critical questions** that expose their vulnerabilities. When you create an argument using a scheme, the system automatically surfaces the questions that would challenge that reasoning pattern.

Crucially, all activity in this layer is mediated by **dialogue moves**. You don't just add an argument to the graph—you ASSERT it. Someone else doesn't just attack—they explicitly REBUT or UNDERCUT. This makes the discourse structure as important as the logical structure.

Component	Model	Key Features
Arguments	`Argument`	Premise claims → conclusion claim, scheme reference, confidence
Premises	`ArgumentPremise`	Links claims as premises with role/order
Edges	`ArgumentEdge`	Inter-argument relations: ATTACK (rebut/undercut/undermine), SUPPORT
Schemes	`ArgumentScheme`, `ArgumentSchemeInstance`	Walton schemes with critical questions
Diagrams	`ArgumentDiagram`	AIF graph representation of argument structure
Chains	`ArgumentChain`, `ArgumentChainNode`, `ArgumentChainEdge`	Threaded reasoning with scopes
Scopes	`ArgumentScope`	Hypothetical/counterfactual groupings

Component	Model	Key Features
Dialogue Moves	`DialogueMove`	ASSERT, WHY, GROUNDS, CONCEDE, RETRACT, CLOSE
Commitments	`Commitment`	Participant commitment stores for dialogue tracking

Key Components:

- `AIFArgumentWithSchemeComposer`: Create arguments with scheme selection
- `AIFArgumentsListPro`: Browse/filter arguments with scheme breakdown
- `ArgumentActionsSheet`: Attack menu with rebut/undercut/undermine options
- `SchemeNavigator` / `SchemeSuggester`: Browse and match argumentation schemes
- `ArgumentChainCanvas`: Visual canvas for threaded argument chains
- `DialogueInspector`: View dialogue move history and commitments
- `CommitmentStorePanel`: Track participant commitments

Layer 3: Outputs & Artifacts (Currently Implemented)

Purpose: Compose reasoning into publishable, citable artifacts.

Deliberation should produce durable outputs. When a working group spends months analyzing a policy question, the result shouldn't evaporate when the project ends. It should persist as institutional memory—citable, auditable, and available for future groups facing similar questions.

Layer 3 provides artifact types designed for different publication needs: **Thesis documents** for legal-style structured arguments with prongs and supporting evidence; **TheoryWorks** for longer-form analysis using established frameworks (Deontological, Instrumental Harm, Teleological Consequentialist, Original Position); **KbPages** for knowledge base entries that compose blocks of content with live links to underlying claims and arguments; **Briefs** for generated summaries.

These artifacts don't just reference the underlying reasoning—they link to it. A thesis claim points to the canonical claim it's asserting. A KB block embeds a live view of an argument that updates as the underlying structure evolves.

Component	Model	Key Features
Thesis	`Thesis`, `ThesisProng`, `ThesisProngArgument`	Legal-style structured documents
TheoryWorks	`TheoryWork`	Longform DN/IH/TC/OP frameworks
DebateSheet	`DebateSheet`, `DebateNode`, `DebateEdge`	Visual debate mapping
KbPages	`KbPage`, `KbBlock`	Knowledge base pages with block system
Briefs	`BriefCompiler`	Generated summaries from deliberation content
Glossary	`GlossaryTerm`	Per-deliberation term definitions

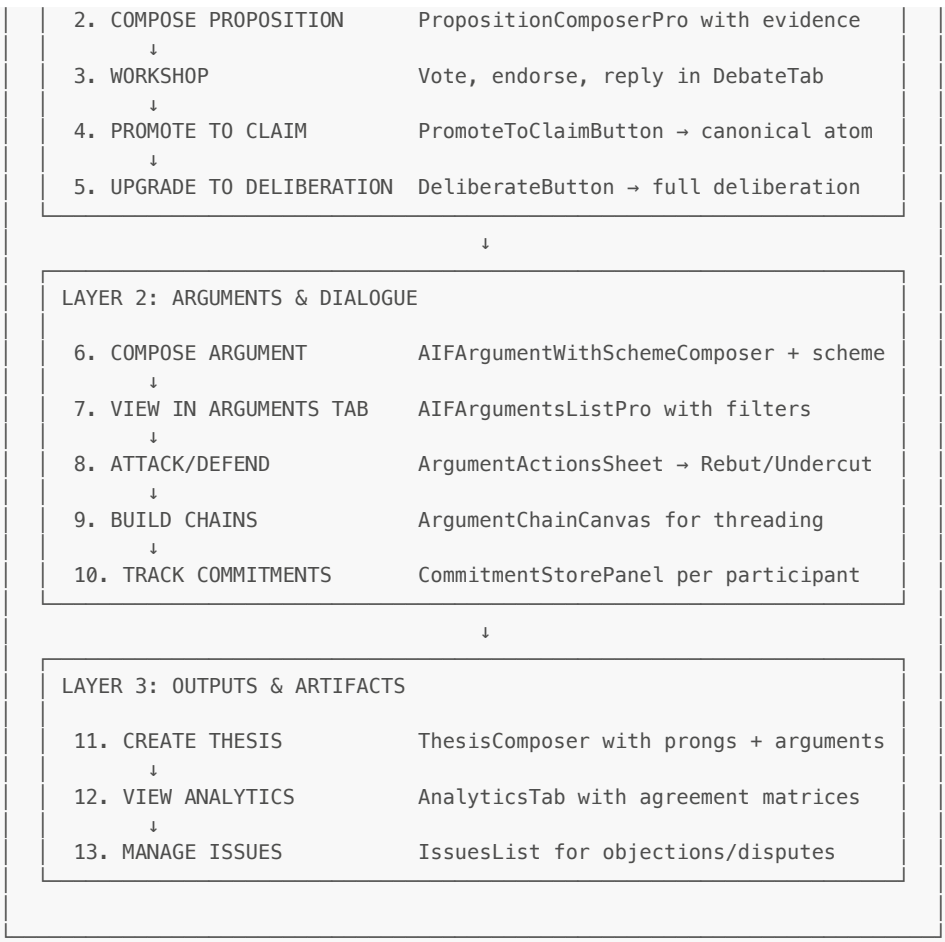
Key Components:

- `ThesisComposer` / `ThesisRenderer`: Create and view structured thesis documents
- `WorksList` / `WorkDetailClient`: Browse and edit TheoryWorks
- `KbPageEditor`: Rich block-based knowledge page editor
- `BriefCompiler`: AI-assisted brief generation
- `DefinitionSheet`: Deliberation glossary management

1.4 The User Journey (Actual Flow)

The three-layer architecture translates into a concrete user journey. A participant might enter at any point—joining an existing deliberation, creating a new proposition, or composing an argument against an existing claim. But the canonical path from scratch illustrates how the layers connect.

The journey below represents the **currently implemented** flow. Every step has working UI components and backend support. This is not a roadmap—it's a description of what the platform does today.



Step	Action	Key Component	Key Model	Status
1	Enter Discussion	`DiscussionView`, `ForumPane`	`Discussion`	✅ Implemented
2	Compose Proposition	`PropositionComposerPro`	`Proposition`	✅ Implemented
3	Workshop	`PropositionsList`	`PropositionVote`, `PropositionEndorsement`	✅ Implemented
4	Promote to Claim	`PromoteToClaimButton`	`Claim` (sourceProposition)	✅ Implemented
5	Upgrade to Deliberation	`DeliberateButton`	`Deliberation` (upgradedFromDiscussionId)	✅ Implemented
6	Compose Argument	`AIFArgumentWithSchemeComposer`	`Argument`, `ArgumentSchemeInstance`	✅ Implemented
7	View Arguments	`AIFArgumentsListPro`	`Argument`, `ArgumentDiagram`	✅ Implemented
8	Attack/Defend	`ArgumentActionsSheet`, `AttackMenuProV2`	`ArgumentEdge`, `DialogueMove`	✅ Implemented
9	Build Chains	`ArgumentChainCanvas`, `ChainsTab`	`ArgumentChain`, `ArgumentChainNode`	✅ Implemented
10	Track Commitments	`CommitmentStorePanel`	`Commitment`	✅ Implemented
11	Create Thesis	`ThesisComposer`	`Thesis`, `ThesisProng`	✅ Implemented
12	View Analytics	`AnalyticsTab`, `CommitmentAnalyticsDashboard`	(computed)	✅ Implemented

Step	Action	Key Component	Key Model	Status
13	Manage Issues	`IssuesList`, `IssueComposer`	`Issue`, `IssueLink`	✅ Implemented

1.5 The DeepDivePanelV2 Hub

Architecture documents often describe systems in terms of modules and data flows, but users experience systems through interfaces. The `DeepDivePanelV2` component is where theory meets practice—where the careful layering of propositions, claims, arguments, and dialogue manifests as clickable tabs and draggable panels.

This single component (~1861 lines in `components/deepdive/DeepDivePanelV2.tsx`) serves as the **central orchestration hub** for all deliberation interfaces. Every deliberation in Mesh renders through this component. Understanding its structure provides the clearest map of what the system actually does, because every feature visible here is a feature that exists.

The hub organizes the full feature set into tabs and floating sheets:

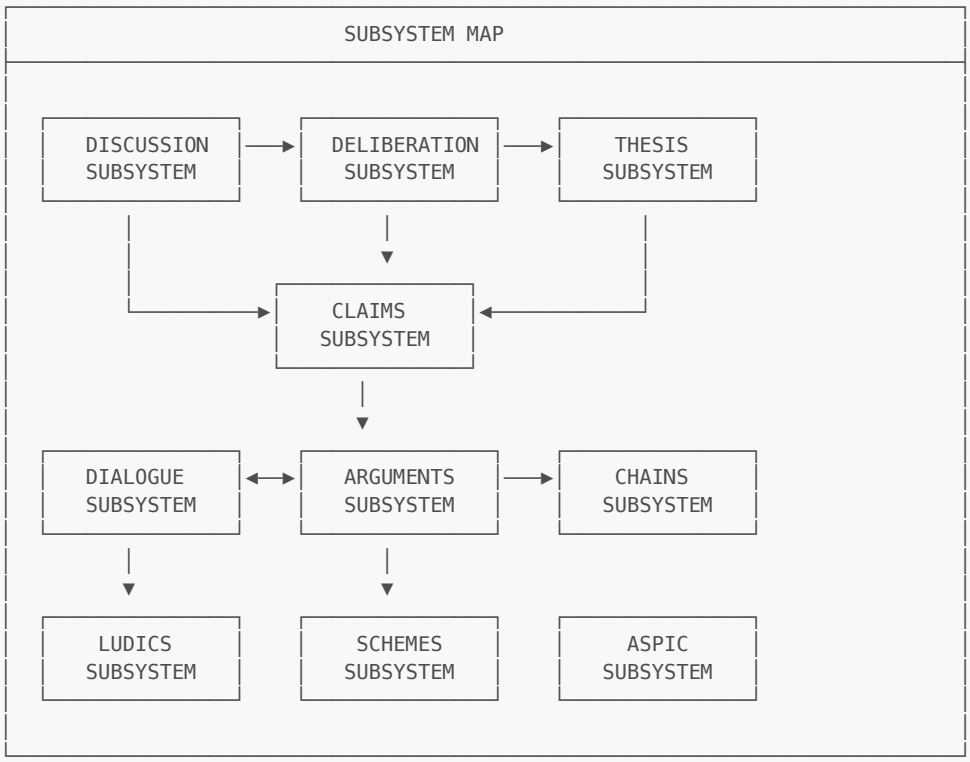
Tab Details:

Tab	Primary Components	Purpose
Debate	`DebateTab`, `PropositionsList`, `ClaimDetailPanel`	Main workspace for claims and propositions
Arguments	`AIFArgumentsListPro`, `SchemeBreakdown`, `AspicTheoryPanel`	Structured argument browsing with scheme analysis
Chains	`ChainsTab`, `ArgumentChainCanvas`	Build threaded argument chains with hypothetical scopes
Ludics	`LudicsPanel`, `BehaviourInspectorCard`	Game-theoretic dialogue analysis
Admin	`DiscourseDashboard`, `IssuesList`, `ActiveAssumptionsPanel`	Moderation and issue tracking
Sources	`EvidenceList`	View and rate evidence sources
Thesis	`ThesisListView`, `ThesisComposer`	Create legal-style structured documents
Analytics	`AnalyticsTab`, `CommitmentAnalyticsDashboard`	Participant agreement matrices

1.6 Key Subsystems Overview

The deliberation platform is not a single application—it is a **federation of subsystems**, each handling a distinct aspect of the reasoning process. These subsystems are deeply interconnected but conceptually separable, allowing developers to understand one domain without mastering all of them simultaneously.

The diagram below shows how subsystems relate. The general flow moves from informal (Discussion) through formal (Arguments, Dialogue) to output (Thesis), with Claims serving as the central currency that all subsystems share. Ludics and ASPIC provide analytical overlays that enrich the core argumentation without blocking basic functionality.



Subsystem	Primary Purpose	Key Models	Key Components
Discussion	Forum/chat conversations	`Discussion`, `ForumComment`, `Conversation`	`DiscussionView`, `ForumPane`
Claims	Canonical atomic assertions	`Claim`, `ClaimEdge`, `ClaimEvidence`	`ClaimMiniMap`, `ClaimDetailPanel`
Arguments	Structured reasoning	`Argument`, `ArgumentPremise`, `ArgumentEdge`	`AIFArgumentsListPro`, `ArgumentActionsSheet`
Dialogue	Move-based discourse	`DialogueMove`, `Commitment`	`DialogueInspector`, `CommitmentStorePanel`
Schemes	Argumentation patterns	`ArgumentScheme`, `CriticalQuestion`	`SchemeNavigator`, `SchemeSuggester`
Chains	Threaded argument flows	`ArgumentChain`, `ArgumentChainNode`, `ArgumentScope`	`ArgumentChainCanvas`, `ChainsTab`
Ludics	Game-theoretic analysis	`LudicDesign`, `LudicAct`	`LudicsPanel`, `BehaviourInspectorCard`
ASPIC	Defeasible reasoning	`ClaimContrary`, extension computation	`AspicTheoryPanel`, `GroundedExtensionPanel`
Thesis	Structured documents	`Thesis`, `ThesisProng`	`ThesisComposer`, `ThesisRenderer`
Deliberation	Core orchestration	`Deliberation`, `AifNode`, `AifEdge`	`DeepDivePanelV2`, `DebateTab`

1.7 Design Principles (Implemented)

Architectural principles only matter if they constrain actual implementation decisions. The principles below are not aspirations—they are constraints that the existing codebase actually satisfies. Each row in the table links a named principle to concrete implementation evidence.

These principles emerged from the theoretical foundations discussed in Section 9 (particularly AIF, Walton schemes, and Girard's Ludics), but they have been operationalized into database columns, API contracts, and UI behaviors. When a principle appears here, it means the system will break if

that principle is violated during development.

Principle	Description	Implementation
AIF Compliance	Argument graphs follow AIF ontology	`AifNode`, `AifEdge` with nodeKind (I/RA/CA/PA/DM)
Dialogue Provenance	Every structure has creator/move tracking	`createdByMoveId`, `introducedByMoveId` on models
Scheme-Based Reasoning	Arguments reference Walton schemes	`ArgumentScheme`, `ArgumentSchemeInstance`, CQ system
Progressive Disclosure	Simple tools first, power on-demand	Discussion → Deliberation, Proposition → Claim
Commitment Tracking	Participant positions are recorded	`Commitment` model, `CommitmentStorePanel`
Attack Taxonomy	Rebut/Undercut/Undermine distinguished	`ArgumentEdge.attackType`, `ArgumentAttackSubtype`
Confidence Quantification	Uncertainty modeled explicitly	Per-argument confidence, DS mode, temporal decay
Composable Outputs	Artifacts build on each other	Thesis references claims/arguments, KB cites deliberations

2. Technical Architecture Overview

Section 1 presented the *conceptual* architecture—what the system does and why. This section presents the *technical* architecture—how the system is actually built, what technologies compose it, and how data flows through its layers.

2.1 Technology Stack

The Agora deliberation system is built on a modern TypeScript stack optimized for real-time collaborative applications:

```
┌─────────────────────────────────────────────────────────────────────────┐
│                          TECHNOLOGY STACK                               │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  PRESENTATION        Next.js 14 (App Router) + React 18 + TypeScript    │
│                      Tailwind CSS, Radix UI, Tiptap (rich text)         │
│                      ReactFlow (graph visualization)                    │
│                                                                         │
│  STATE MANAGEMENT    SWR (data fetching & caching)                      │
│                      React Context + custom hooks                       │
│                      localStorage persistence for UI state              │
│                                                                         │
│  API LAYER           Next.js Route Handlers (app/api/*)                 │
│                      REST conventions with JSON payloads                │
│                                                                         │
│  DATA LAYER          Prisma ORM + PostgreSQL                            │
│                      Supabase (auth, realtime, storage)                 │
│                      Redis (caching, queues via BullMQ)                 │
│                                                                         │
│  EXTERNAL SERVICES   OpenAI (embeddings, generation)                    │
│                      Pinecone (vector search)                           │
│                      LiveKit (real-time communication)                  │
│                      Stripe (payments)                                  │
│                                                                         │
│  INFRASTRUCTURE      Node 18+, Yarn workspaces                          │
│                      AWS (S3, SES, KMS)                                 │
│                      Docker → Kubernetes (production)                   │
│                                                                         │
└─────────────────────────────────────────────────────────────────────────┘
```

2.2 Three-Layer Formal Argumentation Architecture

The technical implementation maps to a **three-layer formal argumentation architecture** drawn from research in computational argumentation. Each layer handles a distinct aspect of structured dialogue:

```
┌─────────────────────────────────────────────────────────────────────────┐
│                      AGORA DELIBERATION ENGINE                          │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  ┌─────────────────────────────────────────────────────────────────┐    │
│  │ LAYER 1: PROTOCOL                                               │    │
│  │ ─────────────────                                               │    │
```

```
┌─────────────────────────────────────────────────────────────────────────┐
│ What moves are legal? What must you do after a challenge?               │
│                                                                         │
│ • Move types: ASSERT, WHY, GROUNDS, CONCEDE, RETRACT, CLOSE             │
│ • Force classification: ATTACK (WHY, GROUNDS), SURRENDER (RETRACT,      │
│   CLOSE, CONCEDE), NEUTRAL (ASSERT, THEREFORE)                          │
│ • Legal move computation based on dialogue state                        │
│ • Commitment tracking (what each participant has asserted/conceded)     │
│                                                                         │
│ Implementation: lib/dialogue/*, app/api/dialogue/legal-moves/*          │
└─────────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
┌─────────────────────────────────────────────────────────────────────────┐
│ LAYER 2: GEOMETRY (Ludics)                                              │
│ ─────────────────────────────────────────────────────────────────────── │
│ Where are we in the dialogue tree? Who has the initiative?              │
│                                                                         │
│ • Loci: Addresses in the interaction tree (e.g., "0.1.2.3")             │
│ • Polarity: P (positive/proponent) vs O (opponent/questioner)           │
│ • Acts: PROPER (regular move) vs DAIMON (†, convergence signal)         │
│ • Designs: Player strategies (what moves are available at each locus)   │
│ • Traces: Records of actual interactions (sequence of acts)             │
│ • Travel status: ONGOING, CONVERGENT (†), DIVERGENT                     │
│                                                                         │
│ Implementation: LudicDesign, LudicAct, LudicLocus, LudicTrace           │
│                 app/api/ludics/*, components/ludics/*                   │
└─────────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
┌─────────────────────────────────────────────────────────────────────────┐
│ LAYER 3: CONTENT (AIF + Walton + ASPIC+)                                │
│ ─────────────────────────────────────────────────────────────────────── │
│ What are the arguments? How do they relate? What's defensible?          │
│                                                                         │
│ • AIF nodes: I (information), RA (inference), CA (conflict),            │
│   PA (preference), DM (dialogue move)                                   │
│ • AIF edges: premise, conclusion, conflictingElement, repliesTo         │
│ • Argumentation schemes (Walton): Expert opinion, Analogy,              │
│   Cause-to-effect, etc. with associated Critical Questions              │
│ • ASPIC+ attack types: REBUT (attack conclusion), UNDERCUT (attack      │
│   inference), UNDERMINE (attack premise)                                │
│ • Grounded semantics: IN / OUT / UNDEC labeling for arguments           │
│ • ClaimContrary: Explicit contrary relations for defeasible logic       │
│                                                                         │
│ Implementation: AifNode, AifEdge, ArgumentScheme, ClaimContrary         │
│                 app/api/aif/*, components/arguments/*                   │
└─────────────────────────────────────────────────────────────────────────┘
```

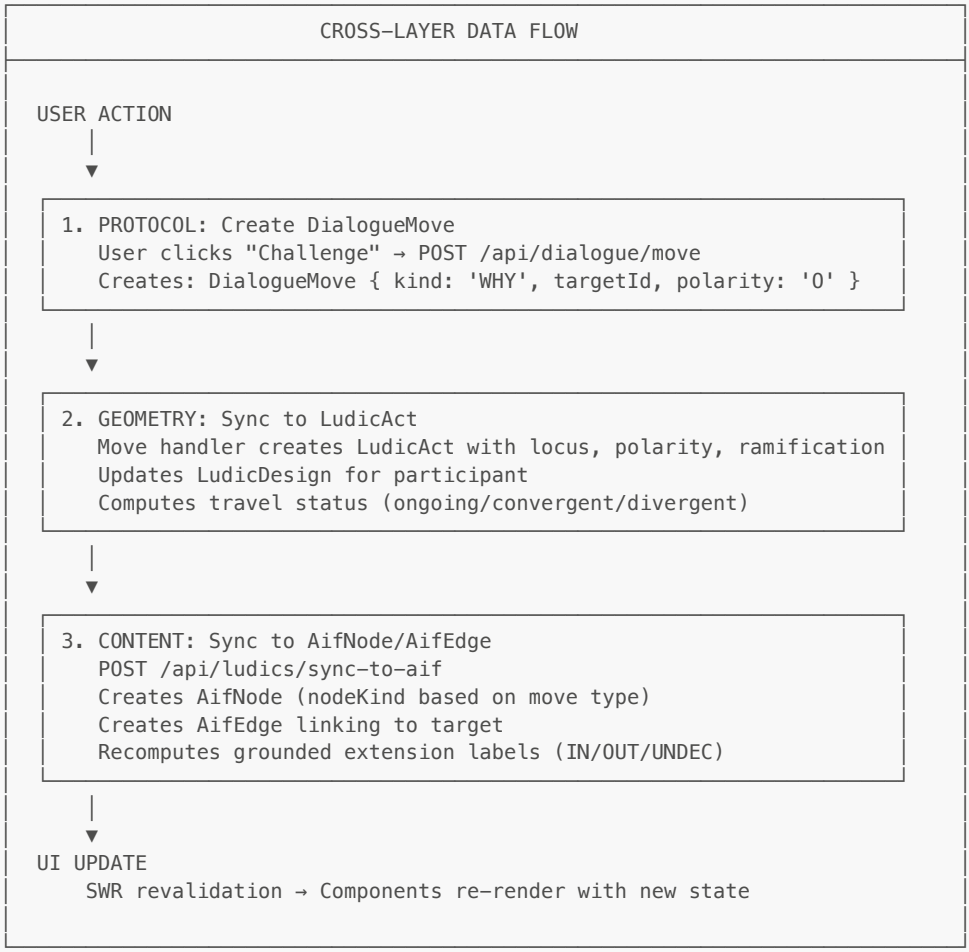
2.3 Layer Mappings to Codebase

Each theoretical layer maps to specific subsystems and database models:

Theoretical Layer	Subsystem	Key Models	Key API Endpoints	Key Components
Protocol	Dialogue	`DialogueMove`, `Commitment`, `DialogueVisualizationNode`	`/api/dialogue/legal-moves`, `/api/dialogue/move`, `/api/dialogue/commitments`	`DialogueActionsButton`, `CommandCard`, `DialogueInspector`, `CommitmentStorePanel`
Geometry	Ludics	`LudicDesign`, `LudicAct`, `LudicLocus`, `LudicTrace`	`/api/ludics/designs`, `/api/ludics/acts`, `/api/ludics/step`, `/api/ludics/sync-to-aif`	`LudicsPanel`, `LociTreeWithControls`, `BehaviourInspectorCard`, `TraceRibbon`
Content	Arguments/Claims/Schemes	`AifNode`, `AifEdge`, `Argument`, `Claim`, `ArgumentScheme`, `ClaimContrary`	`/api/aif/graph`, `/api/aif/schemes`, `/api/arguments/*`, `/api/claims/*`	`AIFArgumentsListPro`, `AifDiagramViewerDagre`, `SchemeBreakdown`, `AspicTheoryPanel`

2.4 Cross-Layer Data Flow

The three layers are not independent—they interact through well-defined synchronization paths:

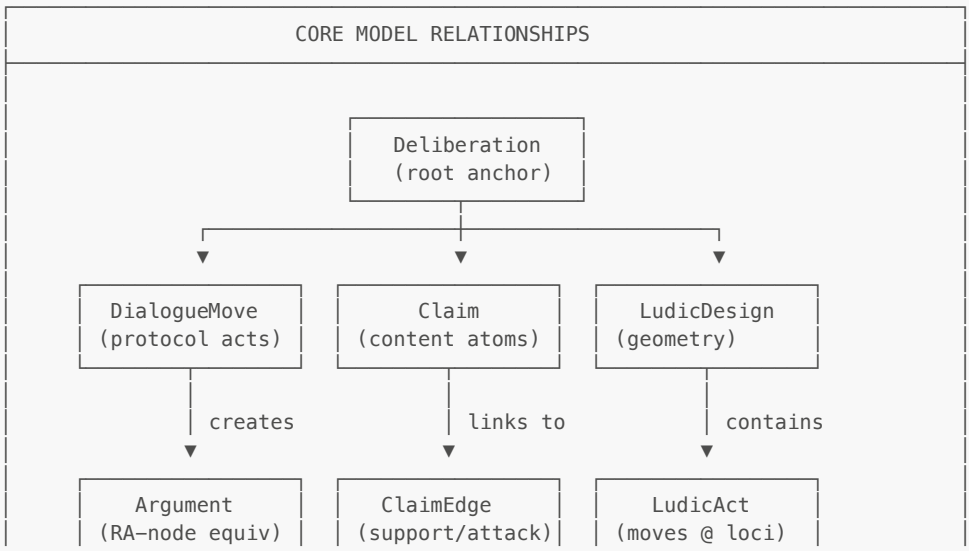


Key Synchronization Endpoints:

- `/api/ludics/sync-to-aif`: Syncs LudicAct rows to AifNode/AifEdge for visualization
- `/api/aif/evaluate`: Recomputes grounded semantics labels after graph changes
- `/api/dialogue/commitments`: Derives commitment stores from DialogueMove history

2.5 Database Model Relationships

The core models form a connected graph with clear provenance chains:



```
│     └──────────┬──────────┘  └──────────┬──────────┘  └──────────┬──────────┘           │
│                │                        │                        │                      │
│                │ syncs to               │                        │ syncs to             │
│                ▼                        ▼                        ▼                      │
│     ┌──────────────────────────────────────────────────────────────────┐                │
│     │                        AifNode / AifEdge                         │                │
│     │          (unified graph representation for export)               │                │
│     └──────────────────────────────────────────────────────────────────┘                │
│                                                                                         │
│  Provenance tracking:                                                                   │
│  • Argument.createdByMoveId → DialogueMove                                              │
│  • Claim.introducedByMoveId → DialogueMove                                              │
│  • AifNode.dialogueMoveId → DialogueMove                                                │
│  • AifNode.ludicActId → LudicAct                                                        │
│  • AifEdge.causedByMoveId → DialogueMove                                                │
│                                                                                         │
└─────────────────────────────────────────────────────────────────────────────────────────┘
```

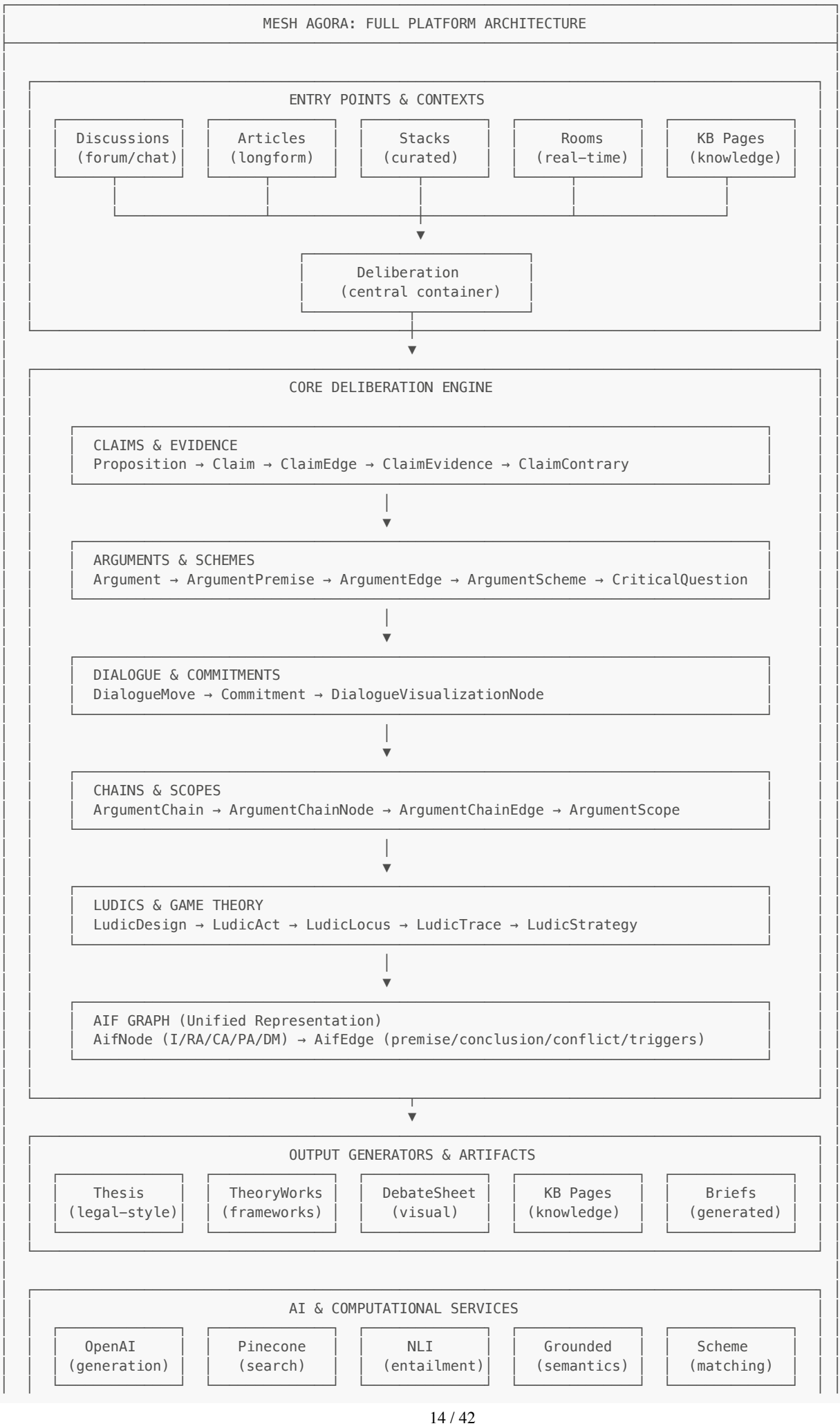
2.6 API Route Organization

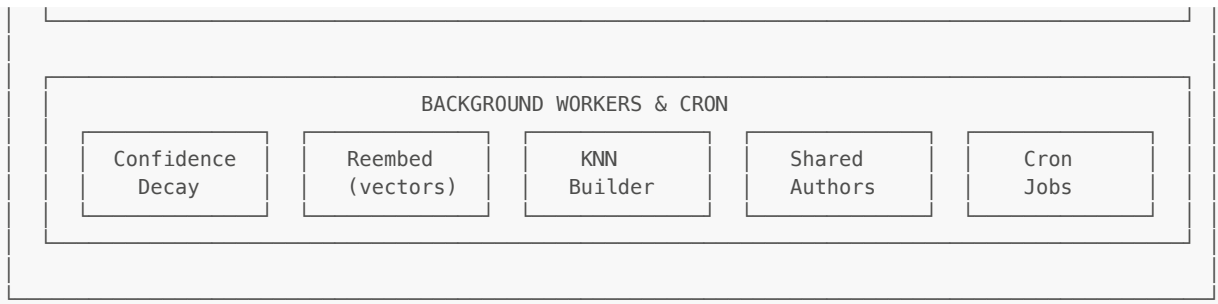
The API layer is organized by domain, with consistent patterns across subsystems:

```
/api
├── dialogue/                    # Protocol layer
│   ├── legal-moves/            # Compute available moves
│   ├── move/                   # Execute a dialogue move
│   ├── commitments/            # Track participant commitments
│   ├── contradictions/         # Detect inconsistencies
│   └── forum/                  # Forum-mode dialogue
│
├── ludics/                      # Geometry layer
│   ├── designs/                # CRUD for LudicDesign
│   ├── acts/                   # CRUD for LudicAct
│   ├── step/                   # Execute interaction step
│   ├── sync-to-aif/            # Sync geometry → content
│   ├── strategies/             # DDS Phase 2: strategy analysis
│   └── correspondences/        # DDS Phase 3: behavior correspondence
│
├── aif/                         # Content layer (AIF)
│   ├── graph/                  # Full AIF graph for deliberation
│   ├── nodes/                  # CRUD for AifNode
│   ├── schemes/                # Argumentation scheme catalog
│   ├── conflicts/              # CA-node management
│   ├── preferences/            # PA-node management
│   ├── evaluate/               # Compute grounded semantics
│   ├── export/                 # AIF-compliant export
│   └── validate/               # Graph validation
│
├── arguments/                   # Argument CRUD
│   ├── [id]/                   # Individual argument
│   ├── diagram/                # Argument diagram generation
│   └── batch/                  # Bulk operations
│
├── claims/                      # Claim CRUD
│   ├── [id]/                   # Individual claim
│   ├── edges/                  # ClaimEdge relations
│   └── label/                  # Semantic labels
│
├── schemes/                     # Scheme catalog & matching
├── contraries/                  # ASPIC+ contrary relations
├── thesis/                      # Thesis document generation
└── deliberations/               # Deliberation management
```

2.7 Full Platform Architecture (Beyond the Three Layers)

The three-layer formal argumentation architecture (Protocol → Geometry → Content) describes the *core reasoning engine*. But the full Mesh platform encompasses much more: entry points, output generators, AI services, background workers, and integrations. This section presents the complete end-to-end architecture.


MESH AGORA: FULL PLATFORM ARCHITECTURE
ENTRY POINTS & CONTEXTS
Discussions (forum/chat)
Articles (longform)
Stacks (curated)
Rooms (real-time)
KB Pages (knowledge)
Deliberation (central container)
CORE DELIBERATION ENGINE
CLAIMS & EVIDENCE
Proposition → Claim → ClaimEdge → ClaimEvidence → ClaimContrary
ARGUMENTS & SCHEMES
Argument → ArgumentPremise → ArgumentEdge → ArgumentScheme → CriticalQuestion
DIALOGUE & COMMITMENTS
DialogueMove → Commitment → DialogueVisualizationNode
CHAINS & SCOPES
ArgumentChain → ArgumentChainNode → ArgumentChainEdge → ArgumentScope
LUDICS & GAME THEORY
LudicDesign → LudicAct → LudicLocus → LudicTrace → LudicStrategy
AIF GRAPH (Unified Representation)
AifNode (I/RA/CA/PA/DM) → AifEdge (premise/conclusion/conflict/triggers)
OUTPUT GENERATORS & ARTIFACTS
Thesis (legal-style)
TheoryWorks (frameworks)
DebateSheet (visual)
KB Pages (knowledge)
Briefs (generated)
AI & COMPUTATIONAL SERVICES
OpenAI (generation)
Pinecone (search)
NLI (entailment)
Grounded (semantics)
Scheme (matching)



2.8 Entry Points & Hosting Contexts

A deliberation doesn't exist in isolation—it's always attached to a hosting context. The platform supports multiple entry points, each with its own UX patterns and use cases:

Entry Point	Model	Description	Path to Deliberation
Discussion	`Discussion`	Lightweight forum/chat threads for informal conversation	`DeliberateButton` upgrades to full deliberation
Article	`Article`	Longform content with embedded deliberation anchors	Annotations spawn focused deliberations
Stack	`Stack`, `StackItem`	Curated collections with commentary	Stack items can anchor deliberations
Room	`AgoraRoom`	Real-time spaces for synchronous deliberation	Deliberation is the room's primary content
KB Page	`KbPage`	Knowledge base entries with embedded arguments	Blocks link to underlying deliberations

Deliberation Hosting:

```
model Deliberation {
  hostType  DeliberationHostType  // 'room' | 'article' | 'stack' | 'discussion'
  hostId    String                // ID of the hosting entity

  // Upgrade path from Discussion
  upgradedFromDiscussionId  String?  @unique
  upgradedFromDiscussion    Discussion?
}
```

2.9 Complete Subsystem Inventory

The platform comprises the following subsystems, each with its own models, API routes, and UI components:

Core Reasoning Subsystems

Subsystem	Purpose	Key Models	Key API Routes	Key Components
Deliberation	Container & orchestration	`Deliberation`, `DeliberationRole`, `DeliberationPref`	`/api/deliberations/*`	`DeepDivePanelV2`
Discussion	Lightweight entry point	`Discussion`, `DiscussionMessage`, `ForumComment`	`/api/discussions/*`	`DiscussionView`, `ForumPane`
Claims	Canonical assertions	`Claim`, `ClaimEdge`, `ClaimEvidence`, `ClaimContrary`	`/api/claims/*`	`ClaimMiniMap`, `ClaimDetailPanel`
Propositions	Pre-claim workshopping	`Proposition`, `PropositionVote`, `PropositionEndorsement`	`/api/propositions/*`	`PropositionComposerPro`, `PropositionsList`
Arguments	Structured reasoning	`Argument`, `ArgumentPremise`, `ArgumentEdge`	`/api/arguments/*`	`AIFArgumentsListPro`, `ArgumentActionsSheet`
Schemes	Argumentation patterns	`ArgumentScheme`, `ArgumentSchemeInstance`, `CriticalQuestion`	`/api/schemes/*`, `/api/aif/schemes/*`	`SchemeNavigator`, `SchemeBreakdown`

Subsystem	Purpose	Key Models	Key API Routes	Key Components
Chains	Threaded argument flows	`ArgumentChain`, `ArgumentChainNode`, `ArgumentChainEdge`, `ArgumentScope`	`/api/argument-chains/*`	`ArgumentChainCanvas`, `ChainsTab`
Dialogue	Move-based protocol	`DialogueMove`, `Commitment`, `DialogueVisualizationNode`	`/api/dialogue/*`	`DialogueInspector`, `CommandCard`
Ludics	Game-theoretic analysis	`LudicDesign`, `LudicAct`, `LudicLocus`, `LudicTrace`	`/api/ludics/*`	`LudicsPanel`, `LociTreeWithControls`
AIF	Graph representation	`AifNode`, `AifEdge`	`/api/aif/*`	`AifDiagramViewerDagre`
ASPIC	Defeasible reasoning	`ClaimContrary` + extension computation	`/api/aspic/*`, `/api/contraries/*`	`AspicTheoryPanel`, `GroundedExtensionPanel`

Output & Artifact Subsystems

Subsystem	Purpose	Key Models	Key API Routes	Key Components
Thesis	Legal-style documents	`Thesis`, `ThesisProng`, `ThesisProngArgument`	`/api/thesis/*`	`ThesisComposer`, `ThesisRenderer`
TheoryWorks	Framework analysis	`TheoryWork`, `TheoryWorkCitation`, `TheoryWorkClaim`	`/api/theory-works/*`	`WorksList`, `WorkDetailClient`
DebateSheet	Visual debate mapping	`DebateSheet`, `DebateNode`, `DebateEdge`	`/api/debates/*`	`DebateSheetCanvas`
KB	Knowledge base	`KbPage`, `KbBlock`, `DebateCitation`	`/api/kb/*`	`KbPageEditor`
Briefs	Generated summaries	(computed)	`/api/briefs/*`	`BriefCompiler`
Glossary	Term definitions	`GlossaryTerm`	`/api/glossary/*`	`DefinitionSheet`
Issues	Objection tracking	`Issue`, `IssueLink`	`/api/issues/*`	`IssuesList`, `IssueComposer`

Supporting Subsystems

Subsystem	Purpose	Key Models	Key API Routes	Key Components
Evidence	Source management	`ClaimEvidence`, `ClaimCitation`	`/api/evidence/*`, `/api/citations/*`	`EvidenceList`
CQ	Critical questions	`CriticalQuestionResponse`, `CqResponseVote`	`/api/cq/*`, `/api/cqs/*`	`CriticalQuestionsV3`
Votes	Endorsement tracking	`PropositionVote`, `ArgumentApproval`	`/api/votes/*`	Vote buttons, approval UI
Analytics	Commitment analytics	(computed from Commitment)	(internal)	`AnalyticsTab`, `CommitmentAnalyticsDashboard`

2.10 AI & Computational Services

The platform integrates several AI and computational services that augment the core reasoning functionality:

```
┌──────────────────────────────────────────────────────────────────────┐
│                    AI & COMPUTATIONAL SERVICES                       │
├──────────────────────────────────────────────────────────────────────┤
│                                                                      │
│  ┌────────────────────────────────────────────────────────────┐      │
│  │ EMBEDDINGS & SEARCH                                        │      │
│  │ • OpenAI embeddings for claims, arguments, evidence        │      │
│  │ • Pinecone vector index for semantic similarity search     │      │
│  │ • Used in: scheme matching, similar claim detection, search│      │
│  │ Implementation: lib/pineconeClient.ts, workers/reembed.ts  │      │
```

```
┌─────────────────────────────────────────────────────────────────────┐
│                                                                     │
│  NATURAL LANGUAGE INFERENCE (NLI)                                   │
│  • Entailment detection between claims                              │
│  • Contradiction detection for ClaimContrary population             │
│  • Configurable threshold (Deliberation.nliThreshold)               │
│  Implementation: lib/nli/*, app/api/nli/*                           │
└─────────────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────────────┐
│  SCHEME SUGGESTION                                                  │
│  • Match claim text to argumentation schemes                        │
│  • Surface critical questions for selected schemes                  │
│  Implementation: lib/schemes/*, app/api/schemes/*                   │
└─────────────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────────────┐
│  GROUNDED SEMANTICS                                                 │
│  • Compute IN/OUT/UNDEC labels for arguments                        │
│  • Dempster-Shafer uncertainty intervals (when dsMode enabled)      │
│  Implementation: lib/semantics/*, app/api/semantics/*               │
└─────────────────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────────────────┐
│  GENERATION & SUMMARIZATION                                         │
│  • Brief generation from deliberation content                       │
│  • Argument generation assistance                                   │
│  • Missing premise suggestion                                       │
│  Implementation: app/api/briefs/*, app/api/compose/*                │
└─────────────────────────────────────────────────────────────────────┘
```

2.11 Background Workers & Scheduled Jobs

Long-running or periodic tasks run outside the request-response cycle via background workers:

Worker	File	Purpose	Trigger
Confidence Decay	`workers/decayConfidenceJob.ts`	Apply temporal decay to stale argument confidence	Cron (daily)
Reembed	`workers/reembed.ts`	Recompute embeddings for modified content	BullMQ queue
User KNN Builder	`workers/user-knn-builder.ts`	Build user similarity graphs	Cron
Shared Author Edges	`workers/computeSharedAuthorEdges.ts`	Link deliberations by shared authors	Cron
Candidate Builder	`workers/candidate-builder.ts`	Build recommendation candidates	Cron
Token Refresh	`workers/tokenRefresh.ts`	Refresh integration tokens	Cron

Worker Infrastructure:

```
// workers/index.ts - Entry point
// Uses BullMQ for job queues, backed by Redis
// Run with: npm run worker
```

Cron Routes (app/api/_cron/):

- `close_auctions/` - Close expired auctions
- Various scheduled maintenance tasks

2.12 State Management Architecture

The frontend manages state at multiple levels:

```
┌─────────────────────────────────────────────────────────────────────┐
│                    STATE MANAGEMENT ARCHITECTURE                    │
├─────────────────────────────────────────────────────────────────────┤
```

```
│                                                                                 │
│   ┌─────────────────────────────────────────────────────────────────────────┐   │
│   │  SERVER STATE (SWR)                                                     │   │
│   │  • useSWR hooks for API data fetching                                   │   │
│   │  • Automatic revalidation on focus, interval, mutation                  │   │
│   │  • Optimistic updates for responsive UI                                 │   │
│   │  Example: useSWR(`/api/deliberations/${id}`)                            │   │
│   └─────────────────────────────────────────────────────────────────────────┘   │
│                                                                                 │
│   ┌─────────────────────────────────────────────────────────────────────────┐   │
│   │  COMPONENT STATE (React hooks)                                          │   │
│   │  • useDeliberationState - tab, config, UI toggles                       │   │
│   │  • useSheetPersistence - floating sheet open/close state                │   │
│   │  • useMinimapData - graph visualization data                            │   │
│   │  Location: lib/hooks/*, components/deepdive/hooks/*                     │   │
│   └─────────────────────────────────────────────────────────────────────────┘   │
│                                                                                 │
│   ┌─────────────────────────────────────────────────────────────────────────┐   │
│   │  PERSISTENT STATE (localStorage)                                        │   │
│   │  • Sheet positions and sizes                                            │   │
│   │  • User preferences                                                     │   │
│   │  • Draft content (propositions, arguments in progress)                  │   │
│   │  Key pattern: dd:sheets:${deliberationId}                               │   │
│   └─────────────────────────────────────────────────────────────────────────┘   │
│                                                                                 │
│   ┌─────────────────────────────────────────────────────────────────────────┐   │
│   │  AUTH STATE (Supabase)                                                  │   │
│   │  • Session management via Supabase SSR                                  │   │
│   │  • User context via AuthContext                                         │   │
│   │  Implementation: lib/AuthContext.ts, lib/supabase-*.ts                  │   │
│   └─────────────────────────────────────────────────────────────────────────┘   │
│                                                                                 │
└─────────────────────────────────────────────────────────────────────────────────┘
```

2.13 Component Directory Structure

The UI is organized into domain-specific component directories:

```
components/
├── deepdive/              # Main deliberation interface
│   ├── DeepDivePanelV2.tsx     # Central orchestration (1861 lines)
│   ├── v3/tabs/                # Tab implementations
│   ├── hooks/                  # Deliberation-specific hooks
│   └── CegMiniMap.tsx          # Claim-evidence graph
│
├── arguments/             # Argument composition & display
│   ├── AIFArgumentsListPro.tsx      # Argument list with filters
│   ├── AIFArgumentWithSchemeComposer.tsx  # Create with scheme
│   ├── ArgumentActionsSheet.tsx     # Attack/defend actions
│   └── SchemeBreakdown.tsx          # Scheme visualization
│
├── claims/                # Claim management
│   ├── ClaimMiniMap.tsx        # Claim relationship graph
│   ├── ClaimDetailPanel.tsx    # Claim inspector
│   └── CriticalQuestionsV3.tsx # CQ interface
│
├── chains/                # Argument chain canvas
│   ├── ArgumentChainCanvas.tsx # Visual canvas editor
│   ├── ChainNode.tsx           # Individual node
│   └── ScopesPanel.tsx         # Scope management
│
├── dialogue/              # Dialogue protocol UI
│   ├── DialogueInspector.tsx   # Move history
│   ├── DialogueActionsButton.tsx # Trigger moves
│   └── command-card/           # Move execution UI
│
├── ludics/                # Game-theoretic views
│   ├── LociTreeWithControls.tsx # Locus tree
│   ├── BehaviourInspectorCard.tsx # Strategy analysis
│   └── TraceRibbon.tsx         # Interaction trace
│
```

```
├── aif/                  # AIF visualization
│   ├── DialogueAwareGraphPanel.tsx
│   └── CommitmentStorePanel.tsx
│
├── aspic/                # ASPIC+ interface
│   ├── AspicTheoryPanel.tsx
│   └── GroundedExtensionPanel.tsx
│
├── thesis/               # Thesis documents
│   ├── ThesisComposer.tsx
│   └── ThesisRenderer.tsx
│
├── propositions/         # Proposition workshopping
│   ├── PropositionComposerPro.tsx
│   └── PropositionsList.tsx
│
├── discussion/           # Discussion/forum
│   ├── DiscussionView.tsx
│   └── ForumPane.tsx
│
├── kb/                   # Knowledge base
│   └── KbPageEditor.tsx
│
└── map/                  # Graph visualizations
    └── Aifdiagramviewerdagre.tsx
```

2.14 Library Directory Structure

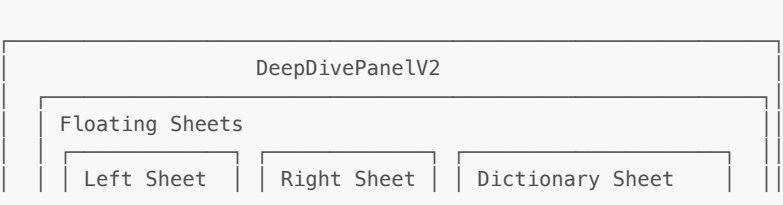
Shared logic and utilities are organized in `lib/`:

```
lib/
├── dialogue/             # Dialogue protocol logic
│   ├── legalMoves.ts          # Legal move computation
│   ├── types.ts               # MoveKind, MoveForce
│   ├── movesToActions.ts      # UI action mapping
│   └── validate.ts            # Move validation
│
├── ludics/               # Ludics computation
│   ├── designs.ts             # Design operations
│   └── traces.ts              # Trace computation
│
├── aif/                  # AIF graph utilities
├── schemes/              # Scheme matching & catalog
├── semantics/            # Grounded semantics
├── nli/                  # Natural language inference
├── chains/               # Chain utilities
│
├── hooks/                # Shared React hooks
├── models/               # Prisma schema
│   └── schema.prisma          # 6989 lines
│
├── prismaclient.ts       # Prisma client singleton
├── redis.ts              # Redis client
├── queue.ts              # BullMQ queue setup
├── pineconeClient.ts     # Pinecone vector client
│
└── supabase-*.ts         # Supabase client variants
```

---

3. Architectural Layers

3.1 Presentation Layer (React/Next.js)



```
│  │ (Explorer)  │ │ (Actions)   │ │ (Glossary)          │  ||
│  └─────────────┘ └─────────────┘ └─────────────────────┘  ||
└──────────────────────────────────────────────────────────┘|
                                                           |
┌──────────────────────────────────────────────────────────┐|
│ Main Tabs                                                ||
│ [Debate][Arguments][Chains][Ludics][Admin][Sources]      ||
│ [Thesis][Analytics]                                      ||
└──────────────────────────────────────────────────────────┘|
└──────────────────────────────────────────────────────────┘
```

3.2 State Management Layer

```
┌──────────────────────────────────────────────────────────┐
│                  State Management                        │
├──────────────────────┬───────────────────────────────────┤
│ Local State          │ SWR Cache                         │
│ • useDeliberationState │ • API Response Caching          │
│ • useSheetPersistence  │ • Optimistic Updates            │
│ • localStorage persist │ • Real-time Revalidation        │
└──────────────────────┴───────────────────────────────────┘
```

3.3 Service/API Layer

```
┌──────────────────────────────────────────────────────────┐
│                    API Routes                            │
├──────────────┬──────────────┬──────────────┬─────────────┤
│ /api/dialogue│ /api/claims  │ /api/arguments│ /api/aif   │
│ • legal-moves│ • [id]       │ • [id]       │ • schemes   │
│ • move       │ • edges      │ • diagram    │ • graph     │
│ • commitments│ • label      │ • batch      │ • validate  │
└──────────────┴──────────────┴──────────────┴─────────────┘
```

3.4 Data Layer (Prisma/PostgreSQL)

```
┌──────────────────────────────────────────────────────────┐
│                  Database Models                         │
├──────────────┬──────────────┬──────────────┬─────────────┤
│ Deliberation │ Claim        │ Argument     │ DialogueMove│
│ Proposition  │ ClaimEdge    │ ArgumentScheme│ Commitment │
│ ArgumentChain│ Evidence     │ CriticalQuestion│LudicDesign│
└──────────────┴──────────────┴──────────────┴─────────────┘
```

---

4. DeepDivePanelV2 - Central Hub

4.1 Component Structure

`DeepDivePanelV2` serves as the **main orchestration component** for deliberation interfaces. Located at `components/deepdive/DeepDivePanelV2.tsx` (~1861 lines).

```
DeepDivePanelV2
├── Props
│   ├── deliberationId: string (required)
│   ├── selectedClaimId?: string
│   ├── hostName?: string
│   └── onClose?: () => void
│
├── State Management
│   ├── useDeliberationState() - Tab, config, UI state
│   ├── useSheetPersistence() - Floating sheet state
│   └── useMinimapData() - Graph visualization data
│
├── Floating Sheets (3)
│   ├── Left: Graph Explorer (Claims/Arguments/Commitments/Analytics)
```

```
│   ├── Right: Actions & Diagram (Dialogical moves, AIF viewer)
│   └── Terms: Definition Dictionary
│
└── Main Tab Content (8 tabs)
    ├── Debate → DebateTab
    ├── Arguments → ArgumentsTab
    ├── Chains → ChainsTab
    ├── Ludics → LudicsPanel
    ├── Admin → Discourse Dashboard, Issues
    ├── Sources → Evidence List
    ├── Thesis → ThesisListView
    └── Analytics → AnalyticsTab
```

4.2 Key Imports & Dependencies

```
// Core UI Components
import { FloatingSheet, SheetToggleButton } from "../ui/FloatingSheet";
import { Tabs, TabsList, TabsContent, TabsTrigger } from "../ui/tabs";

// V3 Tab Components
import { ArgumentsTab, AnalyticsTab, DebateTab } from "./v3/tabs";
import { ChainsTab } from "./v3/tabs/ChainsTab";

// Visualization Components
import { AFMinimap } from '@/components/dialogue/minimap/AFMinimap';
import { AifDiagramViewerDagre } from "@/components/map/Aifdiagramviewerdagre";
import CegMiniMap from "./CegMiniMap";
import ClaimMiniMap from "../claims/ClaimMiniMap";

// Dialogue Components
import { DialogueInspector } from "@/components/dialogue/DialogueInspector";
import { DialogueActionsButton } from "@/components/dialogue/DialogueActionsButton";
import { CommandCard, performCommand } from '@/components/dialogue/command-card/CommandCard';

// AIF Components
import { DialogueAwareGraphPanel } from "@/components/aif/DialogueAwareGraphPanel";
import { CommitmentStorePanel } from "@/components/aif/CommitmentStorePanel";

// ASPIC Components
import { AspicTheoryPanel } from "@/components/aspic/AspicTheoryPanel";

// Ludics Components
import LudicsPanel from "./LudicsPanel";
import BehaviourInspectorCard from '@/components/ludics/BehaviourInspectorCard';
```

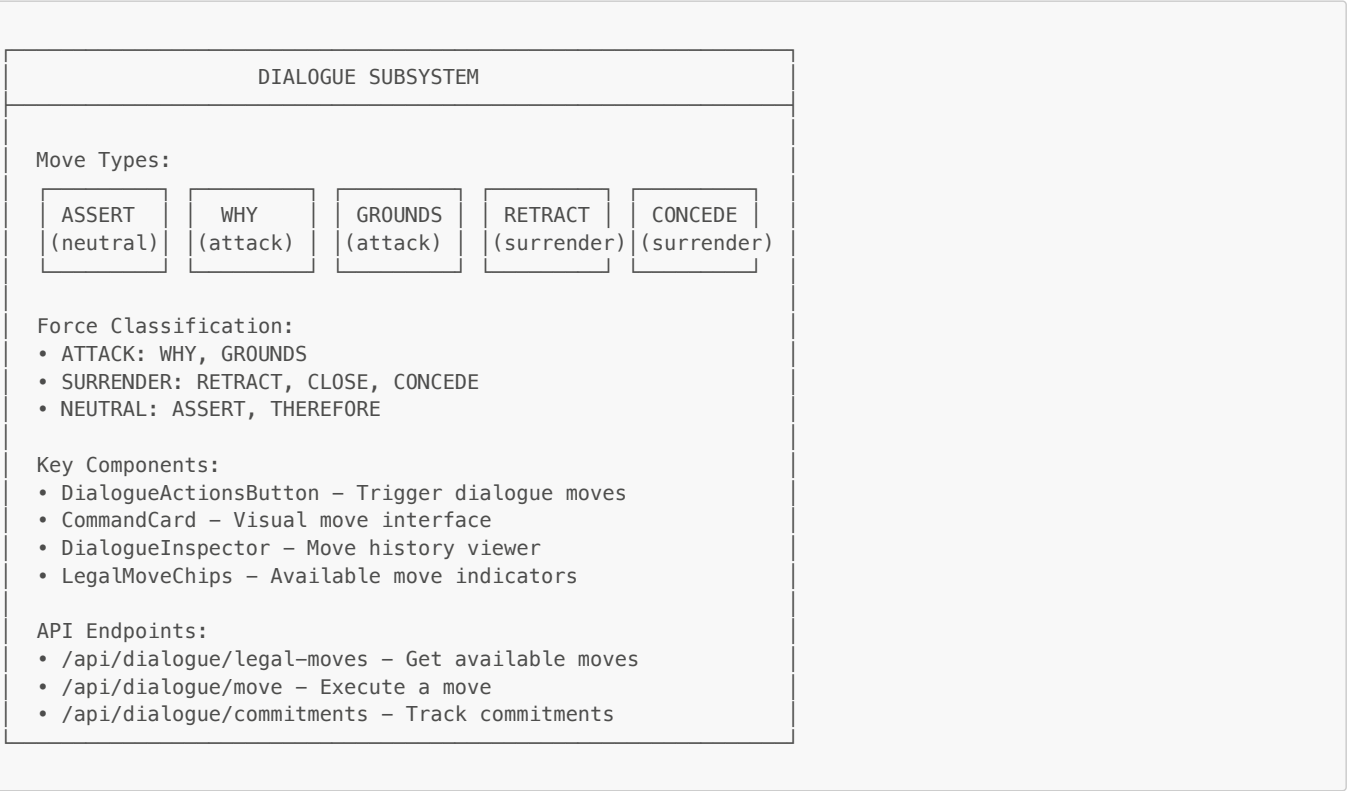
4.3 State Hooks Used

```
// Deliberation state management
const { state: delibState, actions: delibActions } = useDeliberationState({
  initialTab: 'debate',
  initialConfig: { confMode: 'product', rule: 'utilitarian', dsMode: false, cardFilter: 'all' },
});

// Sheet persistence (localStorage-backed)
const { state: sheets, actions: sheetActions } = useSheetPersistence({
  storageKey: `dd:sheets:${deliberationId}`,
  defaultState: { left: false, right: false, terms: false },
});

// Minimap data fetching
const { nodes: minimapNodes, edges: minimapEdges, loading, error } = useMinimapData(deliberationId, {
  semantics: 'grounded',
  supportDefense: true,
  radius: 1,
  maxNodes: 400,
});
```

5. Core Subsystems

5.1 Dialogue Subsystem

Purpose: Manages formal dialogue protocol (PPD moves)

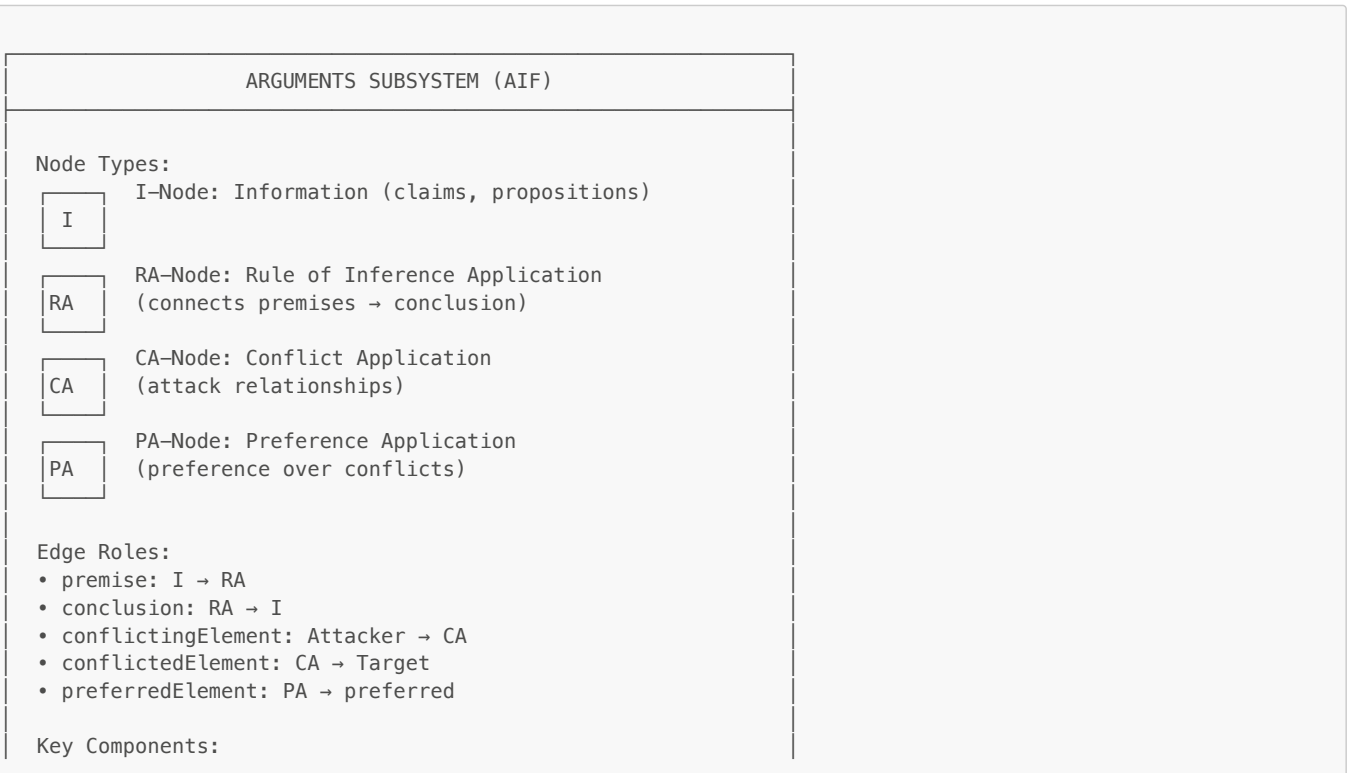


Key Files:

- components/dialogue/DialogueActionsButton.tsx
- components/dialogue/command-card/CommandCard.tsx
- lib/dialogue/legalMoves.ts
- lib/dialogue/types.ts

5.2 Arguments Subsystem (AIF)

Purpose: Structured argumentation using Argument Interchange Format



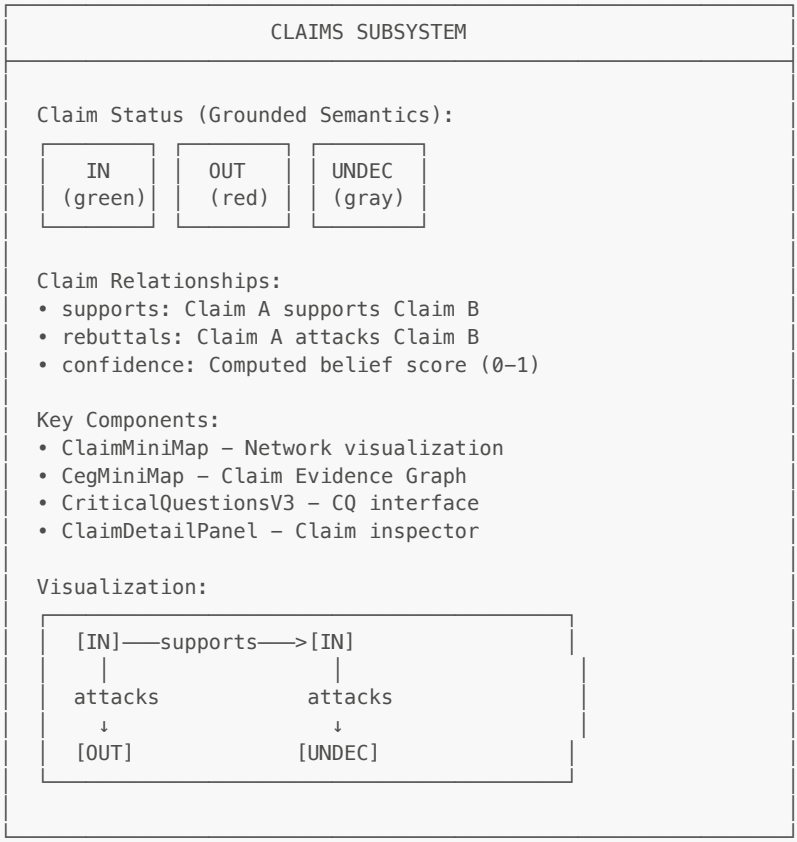
```
│  • AIFArgumentsListPro - Argument list view                    │
│  • AIFArgumentWithSchemeComposer - Create arguments            │
│  • AifDiagramViewerDagre - Graph visualization                 │
│  • SchemeBreakdown - Scheme analysis                           │
│                                                                │
└────────────────────────────────────────────────────────────────┘
```

Key Files:

- components/arguments/AIFArgumentsListPro.tsx
- components/arguments/AIFArgumentWithSchemeComposer.tsx
- components/map/Aifdiagramviewerdagre.tsx
- lib/arguments/diagram.ts

5.3 Claims Subsystem

Purpose: Manage atomic propositions and their evaluation

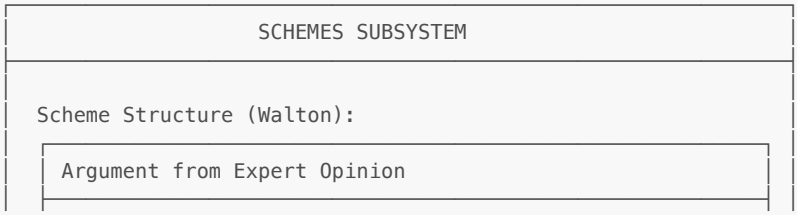


Key Files:

- components/claims/ClaimMiniMap.tsx
- components/claims/CriticalQuestionsV3.tsx
- components/deepdive/CegMiniMap.tsx
- app/api/claims/[id]/route.ts

5.4 Schemes Subsystem

Purpose: Argumentation schemes and critical questions



```
│   │ Premises:                                                    │ │
│   │   P1: E is an expert in domain D                            │ │
│   │   P2: E asserts that A is true                              │ │
│   │   P3: A is within domain D                                  │ │
│   │ Conclusion:                                                 │ │
│   │   C: A is true (presumably)                                 │ │
│   ├─────────────────────────────────────────────────────────────┤ │
│   │ Critical Questions:                                         │ │
│   │   CQ1: How credible is E as an expert?                      │ │
│   │   CQ2: Is E an expert in domain D?                          │ │
│   │   CQ3: What did E actually assert?                          │ │
│   │   CQ4: Is E reliable?                                       │ │
│   │   CQ5: Is A consistent with other experts?                  │ │
│   │   CQ6: Is E's assertion based on evidence?                  │ │
│   └─────────────────────────────────────────────────────────────┘ │
│                                                                   │
│  Attack Types:                                                    │
│  • REBUTS: Attacks the conclusion directly                        │
│  • UNDERCUTS: Attacks the inference rule                          │
│  • UNDERMINES: Attacks a premise                                  │
│                                                                   │
│  Key Components:                                                  │
│  • SchemeBreakdown - Visual scheme structure                      │
│  • SchemeSelector - Scheme picker                                 │
│  • ArgumentCriticalQuestionsModal - CQ interface                  │
│  • SchemeNetBuilder - Multi-scheme networks                       │
│                                                                   │
└───────────────────────────────────────────────────────────────────┘
```

Key Files:

- components/arguments/SchemeBreakdown.tsx
- components/arguments/ArgumentCriticalQuestionsModal.tsx
- app/api/aif/schemes/route.ts
- app/server/services/ArgumentGenerationService.ts

5.5 Ludics Subsystem

Purpose: Game-theoretic analysis of dialogue

```
┌───────────────────────────────────────────────────────────────────┐
│                    LUDICS SUBSYSTEM                               │
├───────────────────────────────────────────────────────────────────┤
│                                                                   │
│  Core Concepts:                                                   │
│  ┌─────────────────────────────────────────────────────────────┐  │
│  │ Locus: Address in dialogue tree (e.g., "0.1.2")             │  │
│  │ Polarity: P (positive/assertive) or O (negative/query)      │  │
│  │ Daimon (†): Convergence marker (dialogue terminates)        │  │
│  └─────────────────────────────────────────────────────────────┘  │
│                                                                   │
│  Act Types:                                                       │
│  • PROPER: Regular move with polarity and locus                   │
│  • DAIMON: Termination signal (†)                                 │
│                                                                   │
│  Travel Status:                                                   │
│  ┌──────────┐ ┌────────────┐ ┌────────────┐                       │
│  │ ONGOING  │ │ CONVERGENT │ │ DIVERGENT  │                       │
│  │(playing) │ │(†,agreed)  │ │(disagreed) │                       │
│  └──────────┘ └────────────┘ └────────────┘                       │
│                                                                   │
│  Design Structure:                                                │
│  ┌──────────────────────────────────────┐                         │
│  │ Proponent Design (pos acts)          │                         │
│  │ ├── Act at locus "0"                 │                         │
│  │ │   └── ramification: ["0.1", "0.2"] │                         │
│  │ └── Act at locus "0.1"               │                         │
│  └──────────────────────────────────────┘                         │
│                                                                   │
│  ┌──────────────────────────────────────┐                         │
│  │ Opponent Design (neg acts)           │                         │
│  │ ├── Act at locus "0" (WHY?)          │                         │
```

```
│  │ └── Act at locus "0.1" (challenge)    │                          │
│  └──────────────────────────────────────┘                          │
│                                                                    │
│  Key Components:                                                   │
│  • LudicsPanel - Main ludics interface                             │
│  • LociTree - Locus visualization                                  │
│  • TraceRibbon - Interaction trace display                         │
│  • BehaviourInspectorCard - Strategy analysis                      │
│                                                                    │
└────────────────────────────────────────────────────────────────────┘
```

Key Files:

- components/deepdive/LudicsPanel.tsx
- components/ludics/LociTreeWithControls.tsx
- packages/ludics-core/types.ts
- packages/ludics-react/LociTree.tsx

5.6 Chains Subsystem (Comprehensive)

Purpose: Link arguments into coherent threads with epistemic scoping, visual canvas editing, and export capabilities.

The Argument Chain system is a core feature of the Agora deliberation platform that allows users to organize, visualize, and export structured arguments. It supports multiple view modes, epistemic status tracking, hypothetical reasoning scopes, and AI-assisted essay generation.

5.6.1 Data Model Architecture

```
┌──────────────────────────────────────────────────────────────────┐
│                    CHAINS DATA MODEL                             │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  ArgumentChain                                                   │
│  ┌──────────────────────────────────────────────────────┐        │
│  │ id: string (cuid)                                    │        │
│  │ name: string                                         │        │
│  │ description?: string                                 │        │
│  │ purpose?: string                                     │        │
│  │ deliberationId: string (FK → Deliberation)           │        │
│  │ createdBy: BigInt (FK → User)                        │        │
│  │ chainType: SERIAL | LINKED | CONVERGENT | DIVERGENT  │        │
│  │ isPublic: boolean                                    │        │
│  │ createdAt: DateTime                                  │        │
│  │ updatedAt: DateTime                                  │        │
│  └──────────────────────────────────────────────────────┘        │
│                          │                                       │
│          ┌───────────────┼───────────────┐                       │
│          ▼               ▼               ▼                       │
│  ArgumentChainNode   ArgumentChainEdge   ArgumentScope           │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

ArgumentChainNode - Represents an argument within a chain:

```
┌──────────────────────────────────────────────────────┐
│ ArgumentChainNode                                    │
├──────────────────────────────────────────────────────┤
│ id: string (cuid)                                    │
│ chainId: string (FK → ArgumentChain)                 │
│ argumentId: string (FK → Argument)                   │
│ role: PREMISE | EVIDENCE | CONCLUSION | OBJECTION |  │
│       REBUTTAL | QUALIFIER                           │
│ nodeOrder: Int                                       │
│ contributorId: BigInt (FK → User)                    │
│ positionX: Float (canvas X coordinate)               │
│ positionY: Float (canvas Y coordinate)               │
│ addedAt: DateTime                                    │
│                                                      │
│ // Phase 4: Epistemic Status & Scopes                │
```

```
| epistemicStatus: EpistemicStatus (default: ASSERTED)       |
| scopeId?: string (FK → ArgumentScope)                      |
| dialecticalRole?: DialecticalRole                          |
| targetType?: ATTACK_NODE | ATTACK_EDGE                     |
| targetEdgeId?: string (for edge attacks)                   |
└────────────────────────────────────────────────────────────┘
```

ArgumentChainEdge - Represents relationships between nodes:

```
┌────────────────────────────────────────────────────────────┐
| ArgumentChainEdge                                          |
├────────────────────────────────────────────────────────────┤
| id: string (cuid)                                          |
| chainId: string (FK → ArgumentChain)                       |
| sourceNodeId: string (FK → ArgumentChainNode)              |
| targetNodeId: string (FK → ArgumentChainNode)              |
| edgeType: SUPPORTS | REFUTES | ENABLES | PRESUPPOSES |     |
|           QUALIFIES | EXEMPLIFIES | UNDERCUTTING_ATTACK |  |
|           REBUTTING_ATTACK | UNDERMINING_ATTACK            |
| strength: Float (0.0 - 1.0)                                |
| description?: string                                       |
| createdAt: DateTime                                        |
└────────────────────────────────────────────────────────────┘
```

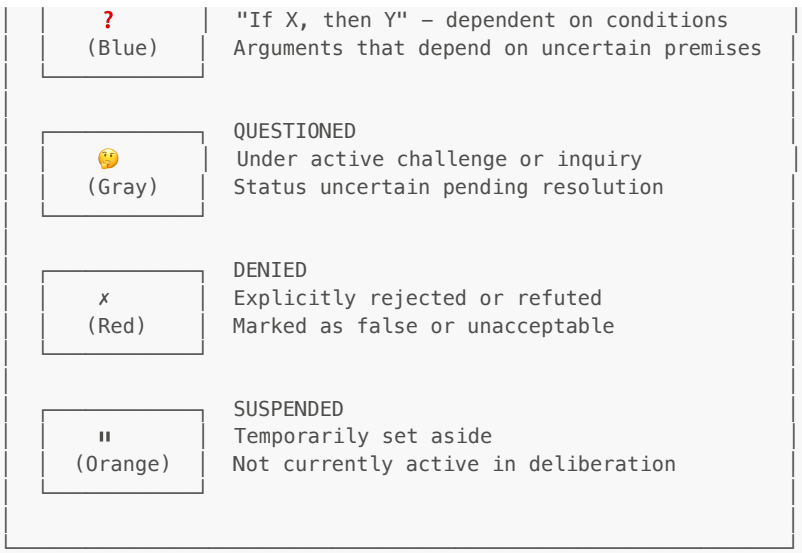
ArgumentScope (Phase 4) - Defines hypothetical reasoning contexts:

```
┌────────────────────────────────────────────────────────────┐
| ArgumentScope                                              |
├────────────────────────────────────────────────────────────┤
| id: string (cuid)                                          |
| chainId: string (FK → ArgumentChain)                       |
| scopeType: HYPOTHETICAL | COUNTERFACTUAL | CONDITIONAL |   |
|            OPPONENT | MODAL                               |
| assumption: string (e.g., "Suppose X...")                  |
| description?: string                                       |
| color?: string (hex color for visual display)              |
| parentScopeId?: string (for nested scopes)                 |
| depth: Int (nesting level, default: 0)                     |
| createdBy: BigInt (FK → User)                              |
| createdAt: DateTime                                        |
└────────────────────────────────────────────────────────────┘
```

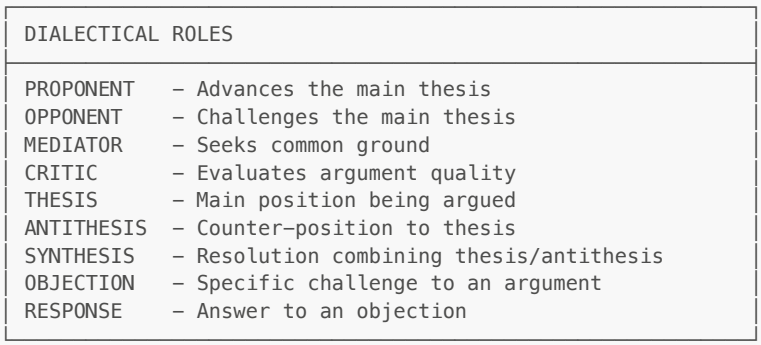
5.6.2 Epistemic Status System (Phase 4)

The epistemic status system tracks the commitment level and modal status of arguments:

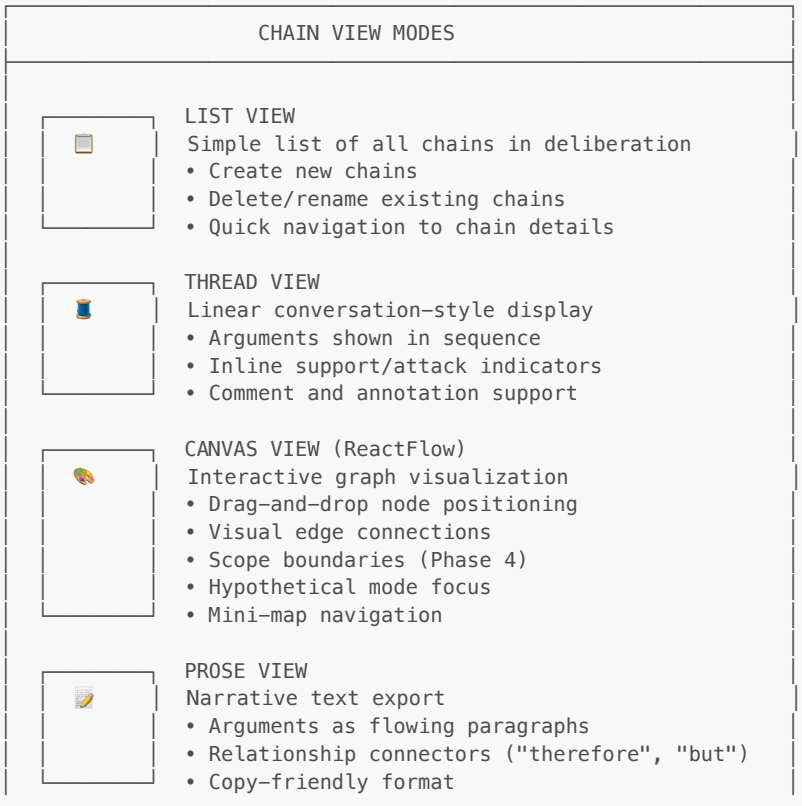
```
┌──────────────────────────────────────────────────────────────┐
|                  EPISTEMIC STATUS VALUES                     |
├──────────────────────────────────────────────────────────────┤
|                                                              |
|  ┌──────────┐  ASSERTED                                      |
|  |    ✓     |  Default status - claimed as true              |
|  | (Green)  |  Represents the "actual world" position        |
|  └──────────┘                                                |
|                                                              |
|  ┌──────────┐  HYPOTHETICAL                                  |
|  |    💡    |  "Suppose X..." - exploring possibilities      |
|  | (Amber)  |  Used within HYPOTHETICAL scopes               |
|  └──────────┘                                                |
|                                                              |
|  ┌──────────┐  COUNTERFACTUAL                                |
|  |    🔮    |  "Had X been the case..." - contrary to fact   |
|  | (Purple) |  Exploring alternative histories               |
|  └──────────┘                                                |
|                                                              |
|  ┌──────────┐  CONDITIONAL                                   |
```



Dialectical Roles - Position within the argument structure:



5.6.3 View Modes



```
│                                                                 │
│   ┌──────────┐  ESSAY VIEW                                      │
│   │          │  AI-generated structured essay                   │
│   │          │  • Introduction, body, conclusion                │
│   │          │  • Citation formatting                           │
│   │          │  • Academic-style output                         │
│   └──────────┘  • PDF export capability                         │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

5.6.4 Canvas Architecture (ReactFlow)

The ArgumentChainCanvas uses ReactFlow for interactive graph editing:

```
┌─────────────────────────────────────────────────────────────────┐
│                CANVAS COMPONENT ARCHITECTURE                    │
└─────────────────────────────────────────────────────────────────┘

  ArgumentChainCanvas.tsx
  ├── ReactFlow Provider
  │   ├── Custom Node Types
  │   │   ├── argument - ChainArgumentNode
  │   │   └── scopeBoundary - ScopeBoundary (Phase 4)
  │   │
  │   ├── Custom Edge Types
  │   │   └── chain - ChainEdge with labels
  │   │
  │   ├── Controls
  │   │   ├── Zoom in/out
  │   │   ├── Fit view
  │   │   └── Fullscreen toggle
  │   │
  │   ├── MiniMap
  │   │   └── Scope-aware coloring
  │   │
  │   └── Background (dots pattern)
  │
  ├── Floating Panels
  │   ├── Add Argument Button
  │   ├── Legend Panel
  │   │   ├── Edge type colors
  │   │   └── Epistemic status indicators
  │   └── Scope Management (Phase 4)
  │
  └── Modal Dialogs
      ├── ChainArgumentComposer - Create new arguments
      ├── EdgeEditor - Edit relationship type/strength
      └── ScopeEditor - Create/edit scopes
```

Node Interaction Flow:

```
┌──────────────┐     ┌──────────────┐     ┌──────────────┐
│  Click Node  │────>│ Show Actions │────>│  Execute     │
│              │     │   Menu       │     │  Action      │
└──────────────┘     └──────────────┘     └──────────────┘
                            │
                ┌───────────┼───────────┐
                ▼           ▼           ▼
           Support      Attack      Edit Node
           (opens       (opens      (opens
           composer)    composer)   detail panel)
```

5.6.5 Scope Boundaries (Phase 4)

Visual containers for hypothetical reasoning:

```
┌─────────────────────────────────────────────────────────────┐
│                SCOPE BOUNDARY RENDERING                     │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│   ┌─────────────────────────────────────────────────────┐   │
│   │ 💡 HYPOTHETICAL: Suppose a $50/ton carbon tax...    │   │
│   │  ┌──────────────────────────────────────────────┐   │   │
│   │  │                                              │   │   │
│   │  │    [Arg 1]──────────────>[Arg 2]             │   │   │
│   │  │       │                     │                │   │   │
│   │  │       └──────────>[Arg 3]<──┘                │   │   │
│   │  │                                              │   │   │
│   │  └──────────────────────────────────────────────┘   │   │
│   │                        [▼ Collapse] [👁 Toggle] [✏ Edit]│   │
│   └─────────────────────────────────────────────────────┘   │
│                                                             │
│  Features:                                                  │
│  • Colored border matching scopeType                        │
│  • Draggable header to reposition                           │
│  • Collapse/expand toggle                                   │
│  • Visibility toggle (dim when not focused)                 │
│  • Click header to enter "Hypothetical Mode"                │
│  • Nodes can be dragged into/out of scopes                  │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Hypothetical Mode:

- When activated, non-scope arguments are dimmed
- New arguments created in this mode auto-assign to the scope
- Composer shows scope context banner
- Visual focus on the hypothetical reasoning thread

5.6.6 Key Components

Component	Location	Purpose
`ChainsTab`	`components/deepdive/v3/tabs/ChainsTab.tsx`	Main tab orchestration
`ChainListPanel`	`components/chains/ChainListPanel.tsx`	List view of all chains
`ArgumentChainCanvas`	`components/chains/ArgumentChainCanvas.tsx`	ReactFlow graph canvas
`ArgumentChainThread`	`components/chains/ArgumentChainThread.tsx`	Linear thread view
`ChainProseView`	`components/chains/ChainProseView.tsx`	Narrative export
`ChainEssayView`	`components/chains/ChainEssayView.tsx`	AI essay generation
`ChainArgumentNode`	`components/chains/ChainArgumentNode.tsx`	Custom ReactFlow node
`ChainArgumentComposer`	`components/chains/ChainArgumentComposer.tsx`	Argument creation dialog
`ScopeBoundary`	`components/chains/ScopeBoundary.tsx`	Visual scope container
`EpistemicStatusBadge`	`components/chains/EpistemicStatusBadge.tsx`	Status indicator

5.6.7 API Endpoints

```
/api/chains/
├── GET    /                        List all chains for deliberation
├── POST   /                        Create new chain
├── GET    /[chainId]               Get chain with nodes and edges
├── PATCH  /[chainId]               Update chain metadata
├── DELETE /[chainId]               Delete chain
│
├── /[chainId]/nodes/
│   ├── POST   /                    Add node to chain
```

```
│   ├── PATCH  /[nodeId]              Update node (position, role, status)
│   ├── DELETE /[nodeId]              Remove node from chain
│   └── PATCH  /[nodeId]/scope        Assign node to scope
│
├── /[chainId]/edges/
│   ├── POST   /                      Create edge between nodes
│   ├── PATCH  /[edgeId]              Update edge type/strength
│   └── DELETE /[edgeId]              Remove edge
│
├── /[chainId]/scopes/
│   ├── GET    /                      List all scopes for chain
│   ├── POST   /                      Create new scope
│   ├── PATCH  /[scopeId]             Update scope
│   └── DELETE /[scopeId]             Delete scope
│
└── /[chainId]/export/
    ├── GET    /prose                 Export as prose text
    └── POST   /essay                 Generate AI essay
```

5.6.8 User Flow: Creating a Chain with Scopes

Step 1: Create a New Chain

```
1. Navigate to Deliberation → Chains Tab
2. Click "New Chain" button
3. Enter:
   - Name: "Carbon Tax Policy Analysis"
   - Description: "Examining carbon pricing policy options"
   - Purpose: "To evaluate arguments for and against carbon taxation"
4. Click "Create"
```

Step 2: Add Main Arguments (ASSERTED)

```
1. Click "+" button or empty canvas area
2. Composer opens with default ASSERTED status
3. Enter argument:
   - Conclusion: "Climate change requires immediate policy action"
   - Add premises from existing claims or create new
   - Select argumentation scheme (e.g., Expert Opinion)
4. Click "Add to Chain"
5. Repeat for other main arguments
```

Step 3: Create Hypothetical Scope

```
1. Click "Add Scope" button in canvas toolbar
2. Configure scope:
   - Type: HYPOTHETICAL
   - Assumption: "Suppose a $50/ton carbon tax is enacted in 2025"
   - Color: Amber (#f59e0b)
3. Click "Create Scope"
4. A visual boundary appears on canvas
```

Step 4: Add Arguments Within Scope

```
Option A: Click inside scope boundary
- Composer auto-fills scope context
- Status defaults to HYPOTHETICAL

Option B: Use Hypothetical Mode
1. Click scope header to activate mode
2. Non-scope arguments dim
3. Create arguments normally
4. All new arguments auto-assign to active scope
```

```
Option C: Drag existing arguments into scope
- Drag node into scope boundary
- Confirmation dialog appears
- Node status updates to match scope type
```

Step 5: Create Edges

```
1. Hover over source node
2. Drag from connection handle to target node
3. Select edge type:
   - SUPPORTS (green)
   - REFUTES (red)
   - QUALIFIES (blue)
   - etc.
4. Optional: Set strength (0-100%)
```

Step 6: Export

```
Prose View:
- Click "Prose" tab
- Copy formatted text

Essay View:
- Click "Essay" tab
- Click "Generate Essay"
- AI produces structured academic essay
- Download as PDF or copy
```

5.6.9 Seed Script Pattern

For testing or bulk data creation, use the seed script pattern:

File: `scripts/seed-test-chain-scopes.ts`

```typescript
/**
 * Seed Script Structure:
 * 1. Configuration - deliberationId, userId, chain metadata
 * 2. Scope definitions - hypothetical contexts
 * 3. Argument definitions - with epistemic status and scope refs
 * 4. Edge definitions - relationships between arguments
 * 5. Execution - create in proper order
 */

// Configuration
const CONFIG = {
  deliberationId: "your-deliberation-id",
  userId: "12",
  chainName: "Your Chain Name",
  chainDescription: "Description...",
  chainPurpose: "Purpose...",
};

// Define scopes
const SCOPES: ScopeData[] = [
  {
    id: "scope-hypothetical-1",
    scopeType: "HYPOTHETICAL",
    assumption: "Suppose X happens...",
    color: "#f59e0b",
  },
];

// Define arguments with epistemic status
const ARGUMENTS: ArgumentData[] = [
  {
    id: "arg-1",
```

```
    conclusionText: "Main thesis...",
    premises: [{ text: "Premise 1" }, { text: "Premise 2" }],
    schemeKey: "practical_reasoning",
    epistemicStatus: "ASSERTED",
    dialecticalRole: "THESIS",
  },
  {
    id: "arg-2",
    conclusionText: "Hypothetical consequence...",
    premises: [{ text: "Given X..." }],
    schemeKey: "causal",
    epistemicStatus: "HYPOTHETICAL",
    scopeRef: "scope-hypothetical-1",  // Links to scope
    dialecticalRole: "THESIS",
  },
];

// Define edges
const EDGES: EdgeData[] = [
  {
    sourceArgId: "arg-1",
    targetArgId: "arg-2",
    edgeType: "ENABLES",
    strength: 0.85,
    description: "Main thesis enables hypothetical exploration",
  },
];

// Execution order:
// 1. Create arguments (with claims)
// 2. Create chain
// 3. Create scopes
// 4. Create nodes (with scope connections)
// 5. Create edges
```

Running the seed script:

```
# Ensure schema is up to date
npx prisma db push
npx prisma generate

# Run seed script
npx ts-node -P tsconfig.scripts.json -r tsconfig-paths/register scripts/seed-test-chain-scopes.ts
```

5.6.10 Key Files Reference

Core Components:

- `components/deepdive/v3/tabs/ChainsTab.tsx` - Tab orchestration
- `components/chains/ArgumentChainCanvas.tsx` - Main canvas (~900 lines)
- `components/chains/ChainArgumentComposer.tsx` - Argument creation dialog
- `components/chains/ChainArgumentNode.tsx` - Custom ReactFlow node
- `components/chains/ScopeBoundary.tsx` - Visual scope container
- `components/chains/EpistemicStatusBadge.tsx` - Status indicator

API Routes:

- `app/api/chains/route.ts` - Chain CRUD
- `app/api/chains/[chainId]/route.ts` - Single chain operations
- `app/api/chains/[chainId]/nodes/route.ts` - Node operations
- `app/api/chains/[chainId]/edges/route.ts` - Edge operations
- `app/api/chains/[chainId]/scopes/route.ts` - Scope operations

Database Schema:

- `lib/models/schema.prisma` - Prisma schema definitions
  - `ArgumentChain` model
  - `ArgumentChainNode` model

- `ArgumentChainEdge` model
- `ArgumentScope` model
- `EpistemicStatus` enum
- `ScopeType` enum
- `DialecticalRole` enum

Seed Scripts:

- `scripts/seed-test-chain.ts` - Basic chain seeding
- `scripts/seed-test-chain-scopes.ts` - Chain with scopes (Phase 4)

5.7 ASPIC Subsystem

Purpose: Formal argumentation framework (ASPIC+)

```
┌─────────────────────────────────────────────────────────────┐
│                    ASPIC+ SUBSYSTEM                         │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  ASPIC+ Components:                                         │
│  • Strict Rules: ψ₁, ..., ψₙ → φ (cannot be attacked)       │
│  • Defeasible Rules: ψ₁, ..., ψₙ ⇒ φ (can be undercut)      │
│  • Contraries: ¬φ is contrary of φ                          │
│  • Preferences: Rule/premise ordering                       │
│                                                             │
│  Attack Types:                                              │
│  ┌─────────────────────────────────────────────────────┐    │
│  │ Rebutting: A attacks B on conclusion                │    │
│  │ Undermining: A attacks B on ordinary premise        │    │
│  │ Undercutting: A attacks B on defeasible rule        │    │
│  └─────────────────────────────────────────────────────┘    │
│                                                             │
│  Extension Semantics:                                       │
│  • Grounded: Unique minimal skeptical extension             │
│  • Preferred: Maximal admissible sets                       │
│  • Complete: All admissible with defended attacks           │
│                                                             │
│  Key Components:                                            │
│  • AspicTheoryPanel - Theory visualization                  │
│  • ConflictResolutionPanel - Resolve conflicts              │
│  • GroundedExtensionPanel - Show grounded set               │
│  • RationalityChecklist - Check argument quality            │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Key Files:

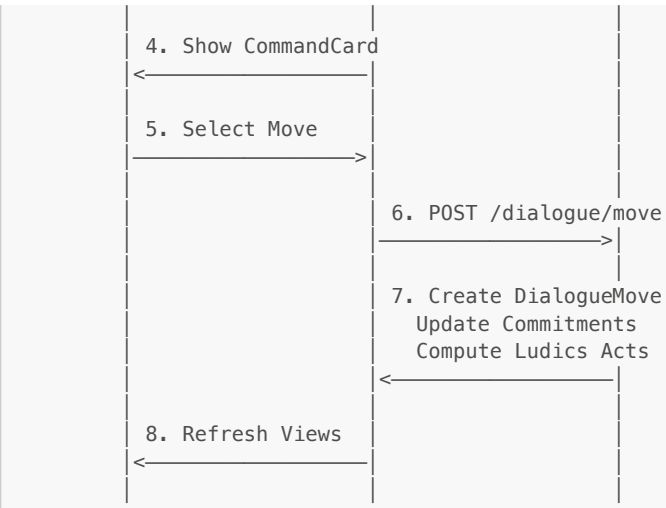
- `components/aspic/AspicTheoryPanel.tsx`
- `components/aspic/ConflictResolutionPanel.tsx`
- `components/aspic/GroundedExtensionPanel.tsx`

---

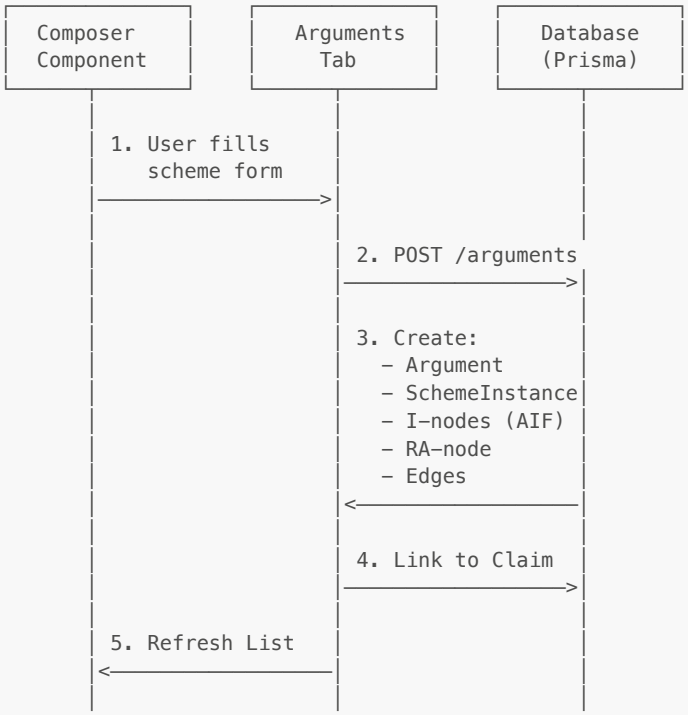
6. Data Flow Diagrams

5.1 Dialogue Move Flow

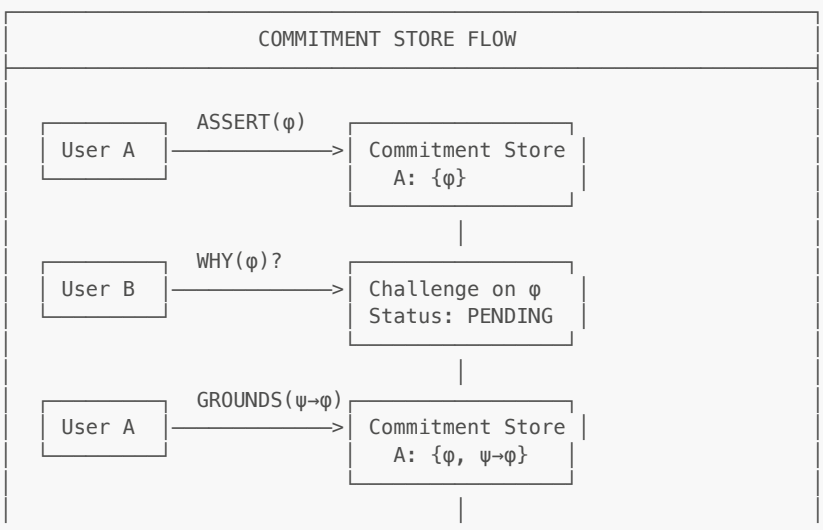
```
┌──────────┐     ┌──────────┐     ┌──────────┐
│   User   │     │DeepDive  │     │ Backend  │
│Interface │     │Panel V2  │     │   API    │
└────┬─────┘     └────┬─────┘     └────┬─────┘
     │                │                │
     │ 1. Click Claim │                │
     │───────────────>│                │
     │                │                │
     │                │ 2. Fetch Legal Moves
     │                │───────────────>│
     │                │                │
     │                │ 3. Return Available
     │                │<───────────────│
     │                │
```



5.2 Argument Creation Flow



5.3 Commitment Tracking Flow



```
│   ┌──────────┐  CONCEDE(φ)  ┌──────────────────┐                  │
│   │ User B   │─────────────>│ Commitment Store │                  │
│   └──────────┘              │   A: {φ, ψ→φ}    │                  │
│                             │   B: {φ}         │                  │
│                             └──────────────────┘                  │
└──────────────────────────────────────────────────────────────────┘
```

---

7. Component Hierarchy

6.1 Full Component Tree

```
DeepDivePanelV2
├── ConfidenceProvider (Context)
│
├── StickyHeader
│   ├── StatusChip
│   ├── ChipBar (Rule, Confidence, DS Mode)
│   ├── Link to /admin/schemes
│   ├── Link to Dialogue Timeline
│   └── DiscusHelpPage
│
├── FloatingSheet [Left] - "Graph Explorer"
│   ├── Tab: Arguments
│   │   └── DialogueAwareGraphPanel
│   │       └── AifDiagramViewerDagre
│   ├── Tab: Claims
│   │   └── CegMiniMap
│   ├── Tab: Commitments
│   │   └── CommitmentStorePanel
│   └── Tab: Analytics
│       └── CommitmentAnalyticsDashboard
│
├── FloatingSheet [Right] - "Actions & Diagram"
│   ├── DialogueActionsButton
│   ├── CommandCard (legacy)
│   └── DiagramViewer
│
├── FloatingSheet [Terms] - "Dictionary"
│   └── DefinitionSheet
│
└── Tabs (Main Content)
    │
    ├── TabContent: "debate"
    │   └── DebateTab
    │       ├── NestedTabs
    │       │   ├── Discussion → ThreadedDiscussionTab
    │       │   ├── Propositions → PropositionComposerPro, PropositionsList
    │       │   ├── Claims → ClaimMiniMap, DialogueInspector
    │       │   └── Sheet View → DebateSheetReader
    │       └── DeliberationSettingsPanel
    │
    ├── TabContent: "arguments"
    │   └── ArgumentsTab
    │       ├── NestedTabs
    │       │   ├── All Arguments → AIFArgumentsListPro
    │       │   ├── Create → AIFArgumentWithSchemeComposer
    │       │   ├── Schemes → SchemesSection
    │       │   ├── Networks → NetworksSection
    │       │   ├── Nets → NetsTab
    │       │   └── ASPIC → AspicTheoryPanel, ConflictResolutionPanel
    │       ├── ArgumentNetAnalyzer (Dialog)
    │       └── AttackArgumentWizard (Dialog)
    │
    ├── TabContent: "chains"
    │   └── ChainsTab
    │       ├── ChainListPanel
    │       ├── ArgumentChainThread
    │       ├── ArgumentChainCanvas
    │       ├── ChainProseView
```

```
      │       └── ChainEssayView
      │
      ├── TabContent: "ludics"
      │   └── LudicsPanel
      │       ├── LociTreeWithControls
      │       ├── LudicsForest
      │       ├── TraceRibbon
      │       ├── JudgeConsole
      │       ├── CommitmentsPanel
      │       └── BehaviourInspectorCard
      │
      ├── TabContent: "admin"
      │   ├── DiscourseDashboard
      │   ├── IssuesList
      │   ├── IssueComposer
      │   ├── CQ Review Dashboard
      │   ├── CreateAssumptionForm
      │   └── ActiveAssumptionsPanel
      │
      ├── TabContent: "sources"
      │   └── EvidenceList
      │
      ├── TabContent: "thesis"
      │   ├── ThesisListView
      │   ├── ThesisComposer (Modal)
      │   └── ThesisRenderer (Modal)
      │
      └── TabContent: "analytics"
          └── AnalyticsTab
```

---

8. API Architecture

7.1 Core API Routes

```
/api/
├── dialogue/
│   ├── legal-moves/         GET  - Available dialogue moves
│   ├── move/                POST - Execute dialogue move
│   ├── move-aif/            POST - AIF-aware move execution
│   ├── commitments/         GET  - Commitment stores
│   ├── contradictions/      GET  - Detect contradictions
│   └── open-cqs/            GET  - Open critical questions
│
├── claims/
│   ├── [id]/
│   │   ├── route.ts         GET/PATCH/DELETE - Claim CRUD
│   │   ├── top-argument/    GET  - Best supporting argument
│   │   ├── edges/           GET  - Claim relationships
│   │   ├── label/           POST - Compute status label
│   │   ├── ca/              POST - Create CA (conflict)
│   │   └── cq/summary/      GET  - CQ status summary
│   ├── batch/               POST - Bulk operations
│   └── search/              GET  - Search claims
│
├── arguments/
│   ├── [id]/
│   │   ├── route.ts         GET/PATCH/DELETE - Argument CRUD
│   │   └── diagram/         GET  - AIF diagram data
│   ├── batch/               POST - Bulk create
│   └── search/              GET  - Search arguments
│
├── aif/
│   ├── schemes/             GET  - All schemes
│   ├── graph-with-dialogue/ GET  - Full AIF graph
│   ├── dialogue/[id]/
│   │   └── commitments/     GET  - Dialogue commitments
│   ├── validate/            POST - Validate AIF structure
│   ├── import/              POST - Import AIF
│   └── export/              GET  - Export AIF
│
```

```
├── deliberations/
│   ├── [id]/
│   │   ├── ludics/          GET/POST - Ludics designs
│   │   ├── viewpoints/      GET/POST - Viewpoint selection
│   │   └── chains/          GET  - Argument chains
│
└── ludics/
    ├── compile/             POST - Compile designs
    ├── interact/            POST - Step interaction
    └── trace/               GET  - Interaction trace
```

7.2 Key API Response Shapes

```typescript
// Legal Moves Response
interface LegalMovesResponse {
  moves: LegalMove[];
  targetId: string;
  targetType: 'claim' | 'argument' | 'proposition';
  locusPath: string;
}

interface LegalMove {
  kind: MoveKind;  // ASSERT, WHY, GROUNDS, RETRACT, CONCEDE, CLOSE
  force: MoveForce; // ATTACK, SURRENDER, NEUTRAL
  targetId: string;
  locusPath: string;
  cqId?: string;
  schemeKey?: string;
  label: string;
  description: string;
  enabled: boolean;
  disabledReason?: string;
}

// AIF Graph Response
interface AIFGraphResponse {
  nodes: AIFNode[];
  edges: AIFEdge[];
}

interface AIFNode {
  id: string;
  kind: 'I' | 'RA' | 'CA' | 'PA';
  label: string;
  schemeKey?: string;
  schemeName?: string;
  dialogueMoveId?: string;
  locutionType?: string;
}

interface AIFEdge {
  id: string;
  from: string;
  to: string;
  role: 'premise' | 'conclusion' | 'conflictingElement' | 'conflictedElement' | 'preferredElement';
}
```

---

9. Theoretical Foundations

8.1 Formal Argumentation Theory

The system implements several formal frameworks:

8.1.1 Abstract Argumentation (Dung)

```
An Abstract Argumentation Framework (AF) is a pair (A, R) where:
- A is a set of arguments
- R ⊆ A × A is an attack relation
```

```
Semantics:
- Conflict-free: S ⊆ A is conflict-free iff ∄a,b ∈ S: (a,b) ∈ R
- Admissible: S is admissible iff S is conflict-free and defends itself
- Grounded: Unique minimal complete extension
```

8.1.2 ASPIC+ Framework

```
ASPIC+ extends AF with:
- Strict rules: X₁, ..., Xₙ → Y (deductive)
- Defeasible rules: X₁, ..., Xₙ ⇒ Y (presumptive)
- Contrariness function: ¯ (maps formulas to contraries)
- Preference ordering: ≺ (over rules and/or premises)

Attack types:
- Rebutting: Argument for ¬φ attacks argument for φ
- Undermining: Argument for ¬ψ attacks premise ψ
- Undercutting: Argument attacks applicability of defeasible rule
```

8.1.3 Argument Interchange Format (AIF)

```
AIF Ontology:
- I-nodes: Information nodes (claims, data)
- S-nodes: Scheme nodes
  - RA-nodes: Rule Application (inference)
  - CA-nodes: Conflict Application (attack)
  - PA-nodes: Preference Application (preference)
  - TA-nodes: Transition Application (dialogue)

Edge types:
- Scheme fulfillment edges (premise, conclusion)
- Support/attack relationships
- Preference relations
```

8.1.4 Ludics (Girard)

```
Key concepts:
- Locus (α): Address in interaction space
- Designs: Sets of chronicles (interaction sequences)
- Polarity: Positive (asserts) / Negative (questions)
- Daimon (†): Termination marker (convergence)

Interaction:
- Two designs interact by matching positive/negative acts
- Convergent: Reaches daimon (agreement)
- Divergent: Blocked (disagreement)
```

8.2 PPD Protocol Rules

```
Move Kinds and Forces:
```

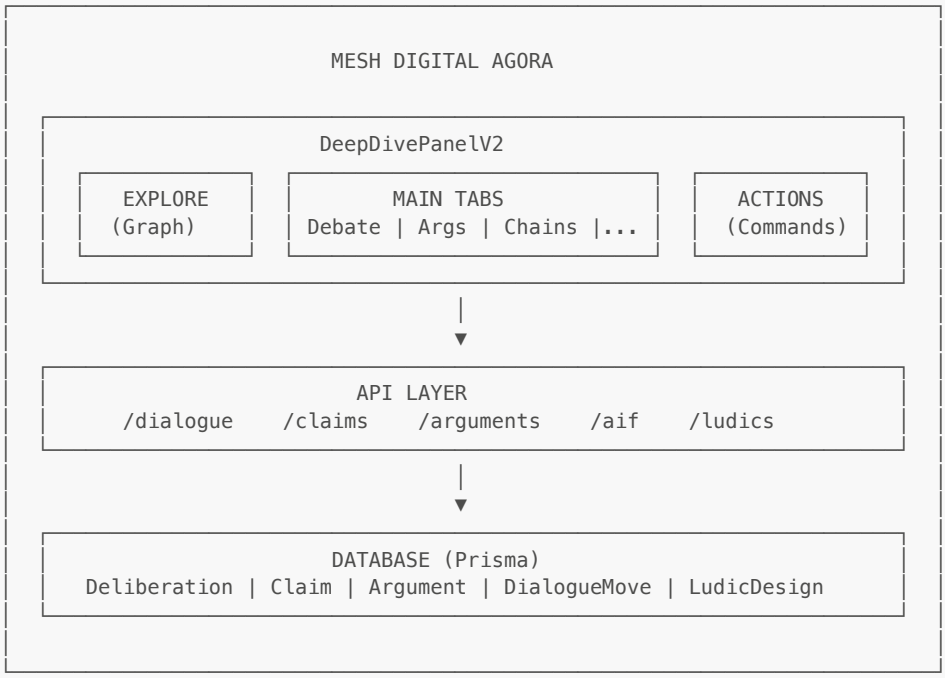
Move Kind	Force	Effect
ASSERT	NEUTRAL	Adds claim to commitment store
WHY	ATTACK	Challenges a commitment
GROUNDS	ATTACK	Provides justification
RETRACT	SURRENDER	Withdraws a commitment
CONCEDE	SURRENDER	Accepts opponent's claim
CLOSE	SURRENDER	Ends branch (daimon †)
THEREFORE	NEUTRAL	Derives conclusion
SUPPOSE	NEUTRAL	Hypothetical assertion
DISCHARGE	NEUTRAL	Exits supposition

```
Key Invariants:
- R4: No duplicate reply to same target/locus/key
- R5: No attacks on surrendered/closed targets
- R7: Cannot concede directly if WHY was answered by GROUNDS
```

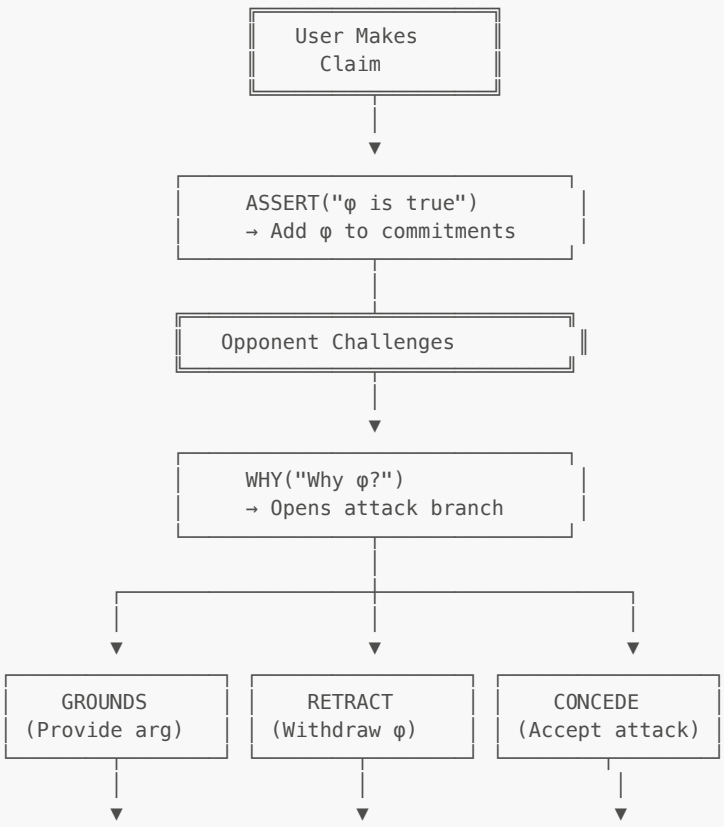
---

10. Whiteboard Diagrams

9.1 System Overview (Simple)

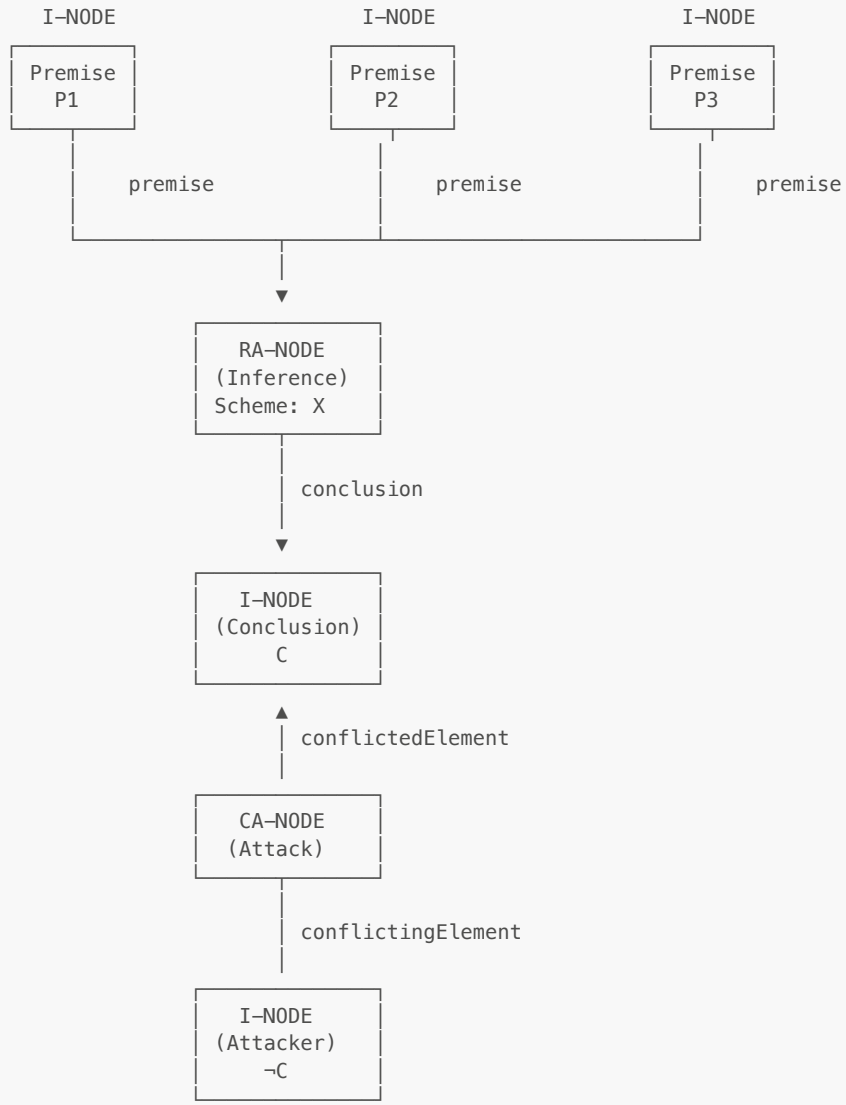


9.2 Dialogue Flow (Whiteboard)

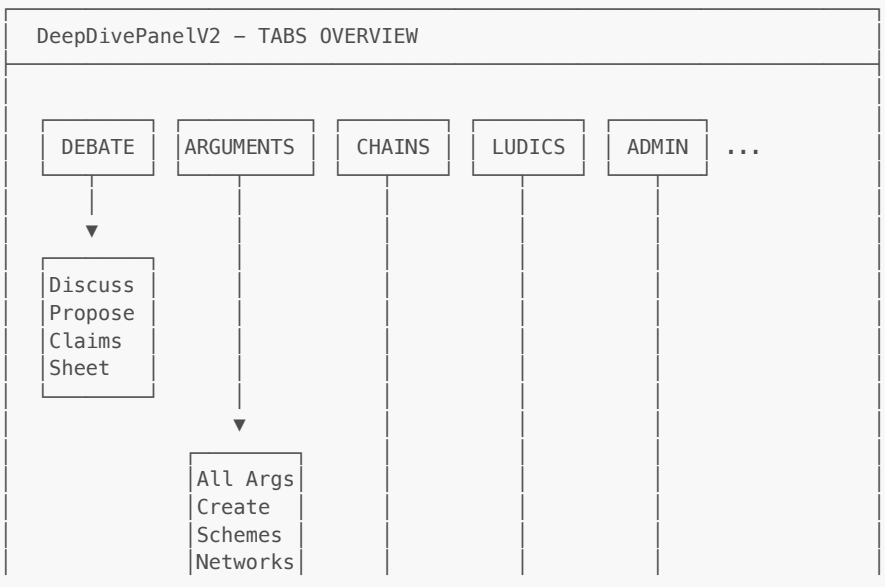


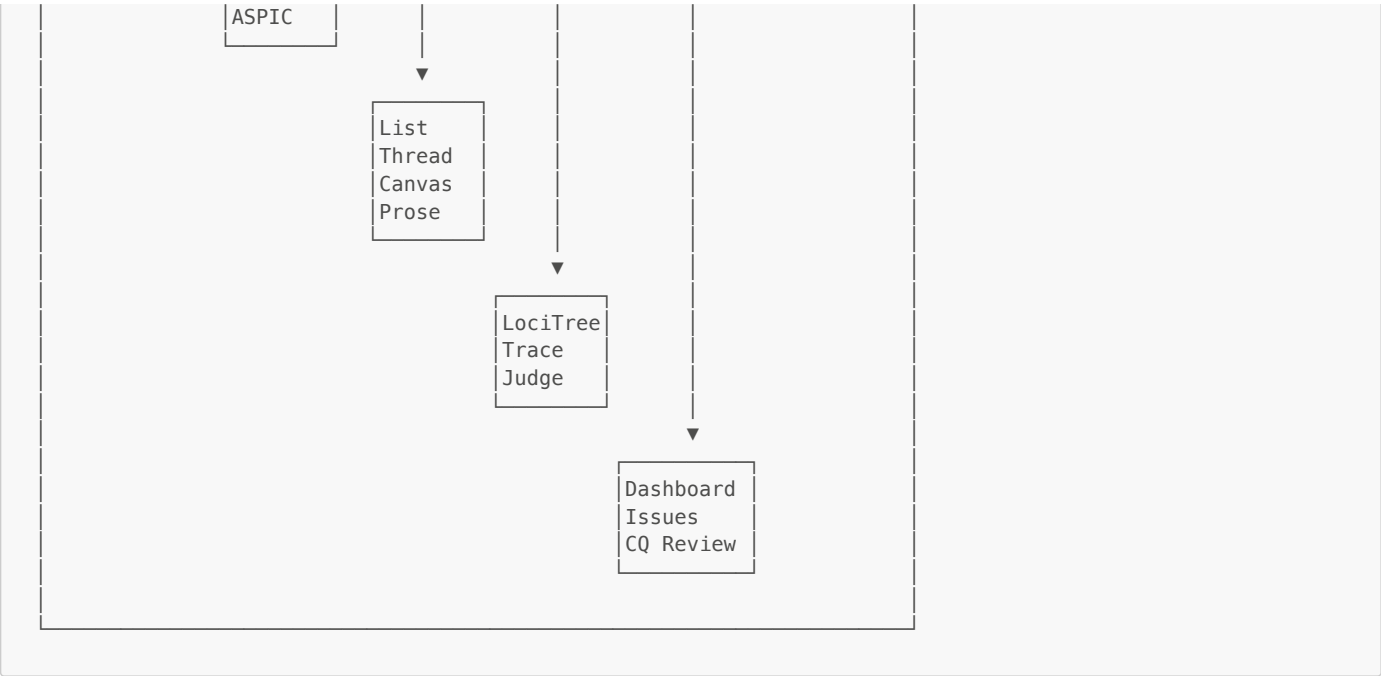
```
    Branch continues    Branch closes       Branch closes
    (more challenges)   (SURRENDER)         (SURRENDER)
```

9.3 AIF Node Structure (Whiteboard)



9.4 Tab Structure (Whiteboard Reference)





Appendix A: File Location Quick Reference

Core Panel

- `components/deepdive/DeepDivePanelV2.tsx` - Main orchestration (~1861 lines)

V3 Tab Components

- `components/deepdive/v3/tabs/DebateTab.tsx`
- `components/deepdive/v3/tabs/ArgumentsTab.tsx`
- `components/deepdive/v3/tabs/ChainsTab.tsx`
- `components/deepdive/v3/tabs/AnalyticsTab.tsx`

V3 Hooks

- `components/deepdive/v3/hooks/useDeliberationState.ts`
- `components/deepdive/v3/hooks/useSheetPersistence.ts`

Dialogue Components

- `components/dialogue/DialogueActionsButton.tsx`
- `components/dialogue/command-card/CommandCard.tsx`
- `components/dialogue/DialogueInspector.tsx`

Argument Components

- `components/arguments/AIFArgumentsListPro.tsx`
- `components/arguments/AIFArgumentWithSchemeComposer.tsx`
- `components/arguments/SchemeBreakdown.tsx`

Visualization

- `components/map/Aifdiagramviewerdagre.tsx`
- `components/claims/ClaimMiniMap.tsx`
- `components/deepdive/CegMiniMap.tsx`

Ludics

- `components/deepdive/LudicsPanel.tsx`
- `packages/ludics-core/types.ts`
- `packages/ludics-react/LociTree.tsx`

ASPIC

- `components/aspic/AspicTheoryPanel.tsx`

- components/aspic/ConflictResolutionPanel.tsx

Libraries

- lib/dialogue/types.ts
- lib/dialogue/legalMoves.ts
- lib/arguments/diagram.ts

Backend Services

- app/server/services/ArgumentGenerationService.ts
- app/server/services/NetIdentificationService.ts

Appendix B: Glossary

Term	Definition
AIF	Argument Interchange Format - standard ontology for argumentation
ASPIC+	Structured argumentation framework with strict/defeasible rules
CA-node	Conflict Application node (attack relationship)
CQ	Critical Question - challenges to argumentation schemes
Daimon (†)	Termination marker in Ludics indicating convergence
Deliberation	A structured discussion/debate instance
Grounded Semantics	Unique minimal skeptical extension of arguments
I-node	Information node (claims, propositions)
Locus	Address in Ludics interaction space (e.g., "0.1.2")
PA-node	Preference Application node
Polarity	P (positive/assertive) or O (negative/questioning)
PPD	Protocol for Persuasion Dialogues
RA-node	Rule Application node (inference relationship)
Scheme	Argumentation pattern (e.g., Argument from Expert)