"Cortex AI"

Tagline: *Your Development Team's AI Brain - Learns, Decides, Automates*

The One-Liner Pitch:

"Cortex AI monitors your entire development workflow, learns from your team's patterns using reinforcement learning, and autonomously makes decisions to optimize your pipeline - from code review to deployment."

Why This WINS

1. Strong Innovation Story

- Not just automation - **autonomous decision-making**
- Not just monitoring - **predictive intelligence**
- Not just AI summaries - **RL-trained on YOUR team's data**

2. Clear Value Proposition

Every developer immediately understands: "An AI that learns my team's patterns and automates decisions? That's game-changing."

3. Perfect Tool Integration (All 5 tools are ESSENTIAL, not forced)

4. Demonstrable Impact

Live demo shows real decisions being made in real-time

5. Scalable Vision

Judges can immediately see how this becomes a real product

THE CORE CONCEPT

What Cortex AI Does:

Think of it as an AI team member that:

1. **Observes** - Monitors your GitHub repo, PRs, CI/CD, deployments
2. **Learns** - Uses RL to understand what "good" looks like for your team
3. **Summarizes** - Provides intelligent insights across your workflow
4. **Decides** - Makes autonomous decisions based on learned patterns
5. **Acts** - Executes decisions through automated workflows

The Magic Moment (What Makes Judges Go "Wow"):

A developer pushes code → Cortex AI analyses it → Learns similar to past successful patterns → Decides to fast-track it → Auto-approves review → Triggers deployment → Monitors success → Updates its decision model.

All autonomous. All learned. All demonstrated live.

HOW EACH TOOL CREATES THE SYSTEM

1. Cline CLI - The Command Interface

Role: Direct interaction layer for developers

Commands You'll Build:

cline analyze *# Analyze current repo state*

cline predict *# Predict PR merge time, deployment risk*

cline decide [action] *# Ask Cortex to make a decision*

cline deploy [env] *# Intelligent deployment with AI checks*

cline learn *# Trigger RL training update*

cline insights *# Get AI-generated insights*

cline status *# Full system status*

Why It's Essential: Gives developers direct access to AI brain. Showcases automation.

---

2. Kestra - The Orchestration Engine

Role: Workflow automation and data pipeline backbone

Workflows You'll Build:

Workflow 1: Data Collection Pipeline

- Continuously pulls data from GitHub (commits, PRs, reviews, CI results)
- Stores in structured format for AI training
- Runs every 15 minutes

Workflow 2: AI Decision Engine

- Triggered on PR creation/update
- Collects contextual data (author history, file changes, test coverage)
- Feeds to Kestra AI Agent
- Executes decisions

Workflow 3: Deployment Orchestration

- Handles staging → production pipeline
- AI-driven deployment decisions
- Rollback triggers based on metrics

Why It's Essential: The nervous system connecting everything. Shows real orchestration complexity.

---

3. Kestra AI Agent - The Decision Maker

BONUS POINTS

Role: Summarizes data AND makes autonomous decisions

Decision Examples:

Decision 1: PR Review Priority

- Summarizes: PR complexity, author reliability, urgency signals
- Decides: "High priority - assign senior reviewer immediately" vs "Low risk - auto-approve after CI"

Decision 2: Deployment Go/No-Go

- Summarizes: Test results, code coverage delta, similar past deployments
- Decides: "Safe to deploy" vs "Hold - similar pattern caused rollback 3 weeks ago"

Decision 3: Resource Allocation

- Summarizes: CI queue length, historical patterns
- Decides: "Allocate extra runners - high traffic period predicted"

Why It's Essential: This gets you BONUS POINTS. Shows AI not just summarizing but actually making calls.

---

4. Oumi - The Learning Brain

Role: RL fine-tuning for personalized intelligence

What You'll Train:

Dataset: Your team's historical data

- Successful PRs vs problematic ones
- Fast deployments vs failed ones
- Good code patterns vs bug-prone patterns

RL Training Goal: Learn to predict outcomes and recommend actions that maximize:

- Code quality

- Deployment success rate
- Developer velocity

The Model Learns:

- "PRs from developer X touching auth code need extra review"
- "Deployments on Friday afternoons have 3x higher rollback rate"
- "Small PRs with high test coverage can be fast-tracked"

Why It's Essential: This is your INNOVATION showcase. Shows real ML, not just API calls.

---

5. Vercel - The Intelligence Dashboard

Role: Real-time visualization and interaction interface

Dashboard Sections:

1. Command Center

- Live workflow status
- Recent AI decisions with reasoning
- System health metrics

2. Intelligence Feed

- Real-time insights stream
- Decision explanations
- Prediction confidence scores

3. Learning Dashboard

- RL training progress
- Model accuracy metrics
- Decision success rate over time

4. Interactive Console

- Chat interface to query Cortex
- Manual decision overrides
- Training data exploration

5. Analytics

- Team velocity trends
- Deployment success rates
- Time saved through automation

Why It's Essential: Makes everything visible. Perfect for demo video. Shows it's a real product.

---

6. CodeRabbit - The Quality Guardian

Role: Code review throughout your development + training data source

How You'll Use It:

During Development (Visible Activity):

- Every feature gets a PR with CodeRabbit review
- Document CodeRabbit suggestions you implement
- Show 10-15 PRs with clear CodeRabbit activity

As Data Source:

- Use CodeRabbit's reviews as training signal for Oumi
- "CodeRabbit flagged security issue" → "Cortex learns to catch similar patterns"
- Makes the system smarter over time

Why It's Essential: Shows development best practices + feeds your AI training.

THE DEMO NARRATIVE (5-7 Minutes)

Act 1: The Problem (1 min)

"Development teams waste 40% of time on decisions that could be automated. Let me show you Cortex AI - an AI that learns your team's patterns and makes decisions for you."

Act 2: The Learning (1.5 min)

- Show Oumi RL training on sample repository data
- Explain: "Cortex learned from 200 PRs, 50 deployments"
- Display model accuracy metrics improving

Act 3: The Intelligence (2 min)

Live Demo:

1. Create a PR in demo repo
2. Show Kestra workflow trigger
3. Kestra AI Agent analyzes and decides
4. Dashboard updates with decision reasoning
5. Run cline predict - see AI predictions
6. Show decision execution

Act 4: The Impact (1.5 min)

- Show analytics: "Cortex saved 15 hours this week"
- Display decision accuracy: 92%
- Explain scalability: "Learns continuously, gets smarter over time"

Act 5: The Vision (1 min)

"Today: PR reviews and deployments. Tomorrow: Sprint planning, incident response, resource optimization. Cortex becomes your team's AI brain."

4-DAY EXECUTION PLAN

DAY 1: Foundation + Quick Win

- Repository setup with proper structure
- Vercel deployment (basic Next.js app)
- CodeRabbit integration enabled
- Kestra Docker setup
- First PR with CodeRabbit review (setup docs)
- Basic dashboard skeleton deployed
- Cline CLI installation and first 2 commands
- First Kestra workflow (GitHub data collection)
- Basic data pipeline working
- 2-3 PRs with CodeRabbit reviews

Deliverable: All tools connected, basic data flowing, deployed dashboard

---

DAY 2: Intelligence Layer

- Oumi environment setup
- Prepare training dataset (can use synthetic + some real data)
- Start RL fine-tuning (let it run while you work on other things)
- Build 3 more Cline commands
- 3-4 PRs with CodeRabbit
- Kestra AI Agent integration
- Build first decision workflow (PR analysis)
- Test decision-making logic

- Dashboard: Add Intelligence Feed section
- 2-3 PRs

Deliverable: AI Agent making first decisions, RL training running, CLI functional

---

DAY 3: Integration + Polish

- Complete Oumi RL training
- Integrate trained model into decision pipeline
- Build deployment decision workflow
- Complete remaining Cline commands
- Dashboard: Add Learning Dashboard section
- 3-4 PRs
- End-to-end testing
- Dashboard: Complete all sections
- Build interactive console
- Polish UI/UX
- 2-3 PRs for bug fixes

Deliverable: Fully integrated system, impressive dashboard, all features working

---

DAY 4: Demo Prep + Submission

- Create demo repository with realistic activity
- Record demo video (multiple takes)
- Write comprehensive README with:
    - Project overview
    - Architecture diagrams
    - Setup instructions
    - Each tool's role explained
    - Screenshots/GIFs
- Final testing

  Polish demo video (editing, captions)
- Final Vercel deployment check
- Verify all CodeRabbit activity visible

- Create submission materials
- Final PR with complete docs
- Submit

Deliverable: Submitted project with killer demo

SUCCESS METRICS

Technical Excellence

- All 5 tools integrated seamlessly
- Working RL model with visible training
- AI making real decisions
- Clean, professional code

Innovation

- Autonomous decision-making (rare in hackathons)
- RL personalization (technical depth)
- Holistic workflow intelligence (big thinking)

Impact

- Solves real pain point
- Quantifiable benefits (time saved, accuracy)
- Clear scalability path

Execution

- Polished dashboard
- Professional demo video
- Comprehensive documentation
- Visible development process (CodeRabbit PRs)

RISK MITIGATION

Risk 1: Oumi RL Training Takes Too Long

Mitigation:

- Use small dataset (200-500 examples)
- Train overnight Day 1 → Day 2
- Have backup: Use pre-trained model with light fine-tuning

Risk 2: Kestra AI Agent Complex

Mitigation:

- Start with simple summarization
- Add decision logic incrementally
- Worst case: 1-2 decisions are enough for demo

Risk 3: Integration Issues

Mitigation:

- Test integrations daily
- Build modularly (each component works standalone)
- Have backup demo data ready

Risk 4: Time Crunch

Mitigation:

- Built-in buffer (plan for 10 hours/day, can go 12 if needed)
- Prioritize core features (marked ⭐ in plan)
- Can cut: Some Cline commands, some dashboard sections

DELIVERABLES CHECKLIST

GitHub Repository

- Clean project structure
- 15+ PRs with CodeRabbit reviews
- Comprehensive README
- Architecture documentation
- Setup instructions
- All source code

Vercel Deployment

- Live dashboard URL
- Stable and responsive
- All features functional
- Professional UI

README

- Project overview with "why"

- Architecture diagram
- Tool integration explanations
- Setup/installation guide
- Usage examples
- Screenshots/GIFs
- Demo video link
- Team information

Demo Video

- 5-7 minutes
- Clear narrative
- Live demonstrations
- Impact metrics
- Professional quality
- Captions/annotations

Tech Stack Breakdown - Cortex AI

MANDATORY TOOLS (Competition Requirements)

1. Cline CLI

- **Domain**: Command-line interface / Developer tools
- **Usage**: Build custom automation commands for the system
- **Tech**: Node.js/TypeScript based CLI tool

2. Kestra

- **Domain**: Backend orchestration / Workflow engine
- **Usage**: Automate workflows, data pipelines, AI agent integration
- **Tech**: YAML workflows, REST API, Docker deployment
- **AI Agent**: Built-in feature for data summarization + decision-making

3. Oumi

- **Domain**: Machine Learning / AI training
- **Usage**: Reinforcement Learning fine-tuning on team's code patterns
- **Tech**: Python library, PyTorch-based

4. Vercel

- **Domain**: Frontend hosting / Deployment platform
- **Usage**: Deploy the dashboard web application
- **Tech**: Next.js deployment

5. CodeRabbit

- **Domain**: Development process / Code review
- **Usage**: PR reviews throughout development, training data source
- **Tech**: GitHub integration

Supporting Technologies

Frontend (Dashboard)

- **Framework**: Next.js 14 (React)
- **Styling**: Tailwind CSS
- **UI Components**: shadcn/ui
- **Charts**: Recharts
- **Deployment**: Vercel
- **Language**: TypeScript

Backend/API

- **API Routes**: Next.js API routes
- **Database**: Optional - PostgreSQL/SQLite (for storing metrics)
- **OR**: Simple JSON file storage
- **Language**: TypeScript/Node.js

Orchestration Layer

- **Kestra**: Workflow orchestration
- **Docker**: Running Kestra instance
- **Language**: YAML (Kestra workflows) + Python (custom tasks)

ML/AI Layer

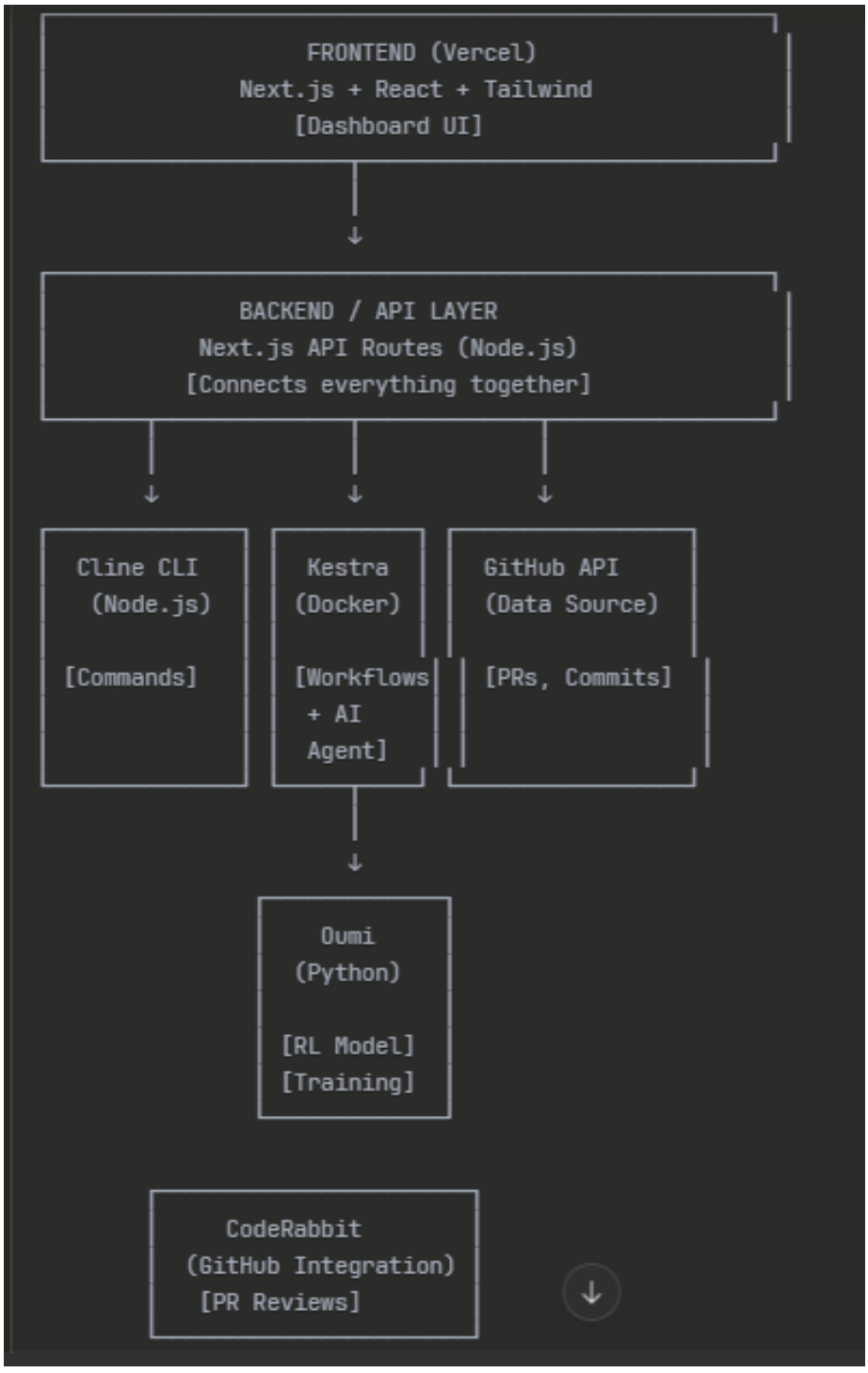
- **Oumi**: RL fine-tuning
- **Python**: Training scripts
- **PyTorch**: Under the hood (via Oumi)
- **Dataset**: JSON/CSV format

CLI Tool

- **Cline CLI**: Command interface
- **Language**: Node.js/TypeScript
- **Integration**: Calls Kestra APIs, GitHub APIs

Data Sources

- **GitHub API**: Pull repo data, PRs, commits
- **REST APIs**: Communication between components



Component	Tool/Tech	Domain	Language
Dashboard	Next.js + Vercel	Frontend	TypeScript
Styling	Tailwind CSS	Frontend	CSS
API	Next.js API Routes	Backend	TypeScript
CLI	Cline	Developer Tool	Node.js
Workflows	Kestra + AI Agent	Orchestration	YAML + Python
ML Training	Oumi	AI/ML	Python
Deployment	Vercel	Hosting	-
Code Review	CodeRabbit	Dev Process	-
Data Source	GitHub API	External	REST
Container	Docker	Infrastructure	-

Efficient Task Division - Cortex AI

TEAM STRUCTURE

Person A (You): Full-Stack + Integration Lead

- **Strengths**: Frontend, Backend, AI
- **Role**: System architect, integration specialist, dashboard builder

Person B (Aditya): AI/ML Specialist

- **Strengths**: AI models
- **Role**: ML training, AI agent logic, intelligent decision-making

PERSON A (YOU) - Responsibilities

Infrastructure & Setup

- GitHub repository setup
- Vercel deployment pipeline
- CodeRabbit integration
- Docker setup for Kestra
- All environment configurations

Frontend Development

- Next.js dashboard (complete UI)
- All dashboard sections:
    - Command Center
    - Intelligence Feed
    - Analytics
    - Interactive Console
- Real-time updates
- Charts and visualizations

Backend/API Development

- Next.js API routes
- GitHub API integration
- Data collection and storage
- API endpoints for dashboard

Cline CLI

- All CLI commands (7 commands)
- CLI → API communication
- Command testing and documentation

Kestra Workflows

- Workflow YAML files
- Data pipeline orchestration
- GitHub data collection workflow
- Deployment workflow
- Integration with AI components

Integration Work

- Connect Oumi model to system
- Connect Kestra AI Agent to workflows
- End-to-end system integration
- Testing all components together

Documentation & Demo

- README file

- Demo video recording
- Architecture diagrams
- Setup instructions

CodeRabbit Management

- Creating all PRs
- Managing reviews
- Ensuring visible activity

PERSON B (TEAMMATE) - Responsibilities

Oumi RL Training (PRIMARY FOCUS)

- Setup Oumi environment
- Prepare training dataset
- Design RL training strategy
- Run RL fine-tuning
- Model evaluation and optimization
- Export trained model
- Create inference code
- Document training process

Kestra AI Agent Logic (SECONDARY FOCUS)

- Design decision-making algorithms
- Create prompts for AI Agent
- Define decision rules and thresholds
- Test AI Agent responses
- Document AI Agent behavior

AI Decision Engine

- PR analysis logic
- Deployment risk assessment
- Prediction algorithms
- Confidence scoring system

Training Data Preparation

- Data cleaning and preprocessing

- Feature engineering
- Dataset formatting

HANDOFF POINTS (Critical Communication)

From Person B → Person A

1. **Day 2 Evening**:
    - Trained model file (.pth or equivalent)
    - Inference code/function
    - Input/output format specification
2. **Day 3 Morning**:
    - Kestra AI Agent prompt templates
    - Decision rules documentation
    - Expected behavior examples
3. **Day 3 Afternoon**:
    - Final model version
    - Performance metrics
    - Edge case handling

From Person A → Person B

1. **Day 1 Evening**:
    - Data format from GitHub API
    - Sample data for training
    - Integration endpoint specs
2. **Day 2 Evening**:
    - Integration API ready
    - Testing environment
    - Kestra workflow structure
3. **Day 3 Morning**:
    - Complete system for AI integration
    - Testing framework
    - Dashboard for visualizing AI decisions

Integration Workflow

- **Modular design** - Each component has clear inputs/outputs
- **API-based communication** - Everything talks via REST APIs
- **Person A handles ALL integration** - You're the integration layer

Progressive Integration:

Day 2 Afternoon:

- You build API endpoints expecting Person B's model (mock it first)
- Test with dummy AI responses

Day 3 Morning (MAIN INTEGRATION):

- Person B gives you: trained model file + inference code
- You integrate model into your API routes (2-3 hours)

Day 3 Afternoon (FINAL INTEGRATION):

- Person B gives you: Kestra AI Agent prompts/logic
- You add to Kestra workflows (1-2 hours)
- Full system testing

Integration Complexity: LOW

1. **Person B delivers 2 simple things:**
    - Python function: predict(data) → decision
    - YAML/prompts: Kestra AI Agent configuration
2. **You just call them:**
    - API route calls Python function
    - Kestra workflow uses AI Agent
3. **No complex dependencies** - Everything is REST/function calls

---

Quick Answer:

- **Where?** Your Next.js API routes
- **When?** Day 3 (morning + afternoon)
- **How long?** 3-4 hours total
- **Easy?** YES - just function calls and API endpoints