# Project "Ask Pete": The Iron Road to Cognitive Sovereignty

## A Phygital Railway Ecosystem for the Future of Learning

### Executive Strategic Synthesis

The "Ask Pete" initiative, formally codified within the Purdue University research ecosystem as **Project Trinity** and the **Daydream Initiative**, represents a paradigmatic shift in the landscape of educational technology. It proposes a fundamental transition from the static, repository-based models of current Learning Management Systems (LMS) to a kinetic, GPS-enabled **"Phygital Railway Ecosystem."** This comprehensive report synthesizes the architectural, pedagogical, and strategic work completed to date, serving as the foundational document for the Purdue Capstone defense, the technical build-out of the "Trinity Server," and the eventual commercial spin-off, **Daydream Labs PBC**.

The contemporary educational landscape is fractured by a binary operational failure known as the **"Edutainment Gap."** On one vector, consumer entertainment platforms utilize high-fidelity mechanics to induce "flow states" and intrinsic motivation but lack rigorous pedagogical structuring. On the opposing vector, traditional institutions rely on LMS platforms like Canvas, which function as digital warehouses—prioritizing content delivery and administrative utility but suffering from "static infrastructure," resulting in engagement attrition and high dropout rates.1

"Ask Pete" addresses this fracture by redefining the educational experience as **"Cognitive Logistics."** In this model, the learner is no longer viewed as a passive vessel to be filled, but as a motive force—a **"Train"**—navigating a complex, non-linear topology of knowledge. The system operationalizes **Cognitive Load Theory (CLT)** into a simulate-able physics engine, treating concepts as "Cargo" with defined mass, and motivation as "Coal," a finite chemical resource. This is not merely gamification; it is **Systems Isomorphism**, where the structural properties of the software architecture—specifically the memory safety of the **Rust** programming language—mirror and reinforce the **Psychological Safety** of the learning environment.2

Technologically, the project leverages a "Bleeding-Edge" stack designed for the post-2025 computing landscape. It utilizes the **Strix Halo** APU to power a **"Hybrid Sovereign"** AI architecture, combining local-first privacy (via **Gemma 3** or **Llama 3** on the NPU/GPU) with cloud-scale orchestration (via **Google Antigravity**). This creates a "Privacy Moat" compliant with FERPA, positioning the platform as a **"Living Laboratory"** for Purdue University under the "Physical AI" partnership with Google.1

The immediate strategic imperative is **Consolidation**. The project currently exists as disparate high-fidelity prototypes: the **"Train Yard"** authoring tool, the **"Node Garden"** gameplay loop, and the **"Weigh Station"** AI constraints. This report culminates in the architectural specification for the **"Trinity Server"** (Grand Central Station), a unified **Rust Modular Monolith** that merges these systems into a single executable, ready for institutional deployment.

## Part I: The Pedagogical Architecture — The Physics of Learning

### 1.1 The Crisis of Static Infrastructure

The prevailing model of digital education relies on the assumption that all learners possess identical "engine specifications"—the same processing speed, working memory capacity, and motivational fuel efficiency. The linear syllabus forces diverse cognitive profiles onto a "Single-Track Railway," creating inevitable bottlenecks. Fast learners are held back by the friction of slow pacing, while slower learners stall on steep gradients of complexity, leading to "derailment" (dropout).

"Ask Pete" posits that knowledge is inherently **rhizomatic**—it grows in non-linear networks, branching and interconnecting in unpredictable ways. A static LMS cannot model this topology. The solution is a **Kinetic Ecosystem** where the infrastructure adapts to the learner's velocity and capacity, transforming the pedagogy from a **"Library" metaphor** (storage and retrieval) to a **"Logistics" metaphor** (transport and delivery).2

### 1.2 The Railway Ecosystem Metaphor: Operationalizing Cognitive Load

The "Railway Ecosystem" is a rigorous translation of cognitive science principles into a simulation engine. It converts abstract psychological variables into tangible physics properties that can be modeled, measured, and optimized.

#### 1.2.1 The Student as the Train

The learner is conceptualized as a physical machine defined by measurable attributes.

* **The Engine (Germane Load):** Represents the student's executive function and processing power. A larger engine can haul heavier concepts but burns fuel at a higher rate.
* **The Chassis (Schema Structure):** Represents the student's prior knowledge. New information ("Cargo") must be securely fused to the chassis; otherwise, it is lost during transit.
* **Velocity (Mastery Rate):** The speed at which a student traverses the curriculum.
* **Locomotive Profiles (Jungian Archetypes):** Instead of traditional RPG classes, students adopt archetypes that influence their starting stats 2:
  + **The Interceptor Express (The Hero):** High speed and combustion. Excellent for "Boss Battles" (exams) but suffers from "Brittle Wheels," making them prone to catastrophic failure under sustained stress.
  + **The Heavy Hauler (The Sage):** Immense cargo capacity. Can transport "Class III" deep theoretical concepts but moves slowly, requiring patience.
  + **The Maintenance-of-Way (The Creator):** gains XP by building tracks for others, utilizing "Terraforming" mechanics to bridge gaps in the curriculum.

#### 1.2.2 Knowledge as Cargo (Intrinsic Load)

In the "Ask Pete" ecosystem, instructional content is assigned a specific **"Cargo Weight"** corresponding to its **Intrinsic Load**.

* **Class I Cargo:** Simple facts and rote memorization (Light).
* **Class II Cargo:** Procedural knowledge and application (Medium).
* **Class III Cargo:** Abstract theoretical concepts and synthesis (Heavy).

The system enforces physics-based constraints. If an instructional designer attempts to load "Class III" cargo onto a train with insufficient engine power, the train will physically stall. This **"Safety Lockout"** prevents Cognitive Overload by making it mathematically impossible within the simulation, forcing the designer to chunk the content into smaller, manageable payloads.1

#### 1.2.3 Motivation as Coal (The Fuel Economy)

Motivation is treated as a finite chemical resource, **"Coal,"** derived from **Self-Determination Theory (SDT)**. It is not an abstraction but a tangible fuel.

* **Mining:** Coal is generated by satisfying the needs for Autonomy (choosing a route) and Competence (overcoming a challenge).
* **Combustion:** Coal is burned to generate **"Steam"** (Progress). The **Combustion Rate** is variable; high-focus tasks ("inclines") burn fuel rapidly.
* **The Economy:** Students must manage their fuel reserves. "Banking" coal allows them to prepare for difficult sections of the track. If a student runs out of coal, the train stops—modeling burnout. The system then intervenes with "Refueling" protocols (lower-stakes engagement or rest) rather than punishment.2

#### 1.2.4 Friction and Terrain (Extraneous and Germane Load)

* **Track Friction (Extraneous Load):** Poorly designed instruction, confusing interfaces, or "social anxiety" act as rust on the rails, increasing resistance. This forces the engine to burn more fuel to maintain the same velocity.
* **Gradients (Desirable Difficulty):** The system intentionally introduces inclines. Climbing a gradient requires increased combustion, generating the heat necessary to "fuse" the cargo to the chassis (Long-Term Memory encoding). A track that is too flat (too easy) fails to generate this heat, resulting in poor retention.1

## Part II: The Technical Architecture — The "Trinity" Stack

The philosophical mandate of "Systems Isomorphism"—where the code mirrors the pedagogy—dictates a specific, high-performance technology stack. The architecture must be **Memory Safe** (Rust), **Local-First** (Privacy), and **Hybrid Sovereign** (Scalable).

### 2.1 The Hardware Substrate: Strix Halo and the Compute Triad

The physical anchor for the "Trinity" system is the **AMD Strix Halo** APU (Ryzen AI Max+ 395), specifically housed in the **GMKtec EVO-X2** workstation. This hardware is selected for its massive **Unified Memory Architecture (UMA)**, which allows the CPU and GPU to share a configured pool of 96GB VRAM (Variable Graphics Memory).4

This enables the **Compute Triad**, assigning cognitive functions to the processor best suited for them:

| **Processor** | **Component** | **Function in "Ask Pete"** |
| --- | --- | --- |
| **CPU** | **Zen 5** (16 Cores) | **Executive Functions:** Runs the OS (Linux Fedora/Arch), orchestration logic (LangGraph), vector database (ChromaDB/LanceDB), and the Rust compiler. The high core count ensures rapid compilation of the "Modular Monolith." |
| **GPU** | **Radeon 8060S** (40 CUs) | **Cognitive Core:** Dedicated exclusively to LLM inference. The 96GB VRAM allows for running massive quantized models like **GPT-OSS 120B** or **Llama 3 70B** locally, providing "cloud-class" intelligence without data exfiltration.4 |
| **NPU** | **XDNA 2** (50 TOPS) | **Sensory Cortex:** Handles always-on, low-power tasks. It runs **Whisper** for speech-to-text and **nomic-embed-text** for generating RAG embeddings, preserving the GPU for deep reasoning tasks.4 |

**Thermal Dynamics:** The system is tuned to a "Sweet Spot" TDP of **80W-90W**. This balances token generation rates with acoustic comfort, ensuring the "Familiar" (the workstation) operates quietly enough to be an unobtrusive presence in the learning environment.4

### 2.2 The System Layer: Linux and Driver Ecosystem

The operating system of choice is **Fedora Workstation** (or a customized Arch build). This "upstream-first" distribution provides the bleeding-edge kernel support (Kernel 6.14+, Mesa 25.0) required for the Strix Halo drivers.4

#### 2.2.1 Acceleration Stack: The Fail-Over Strategy

A critical reliability feature is the **GPU Acceleration Fail-Over Strategy**.

* **Primary Path (ROCm):** The system attempts to initialize the AMD **ROCm 7.0** stack using the kyuz0/amd-strix-halo-toolboxes container. This provides peak performance (approx. 37 TFLOPS).
* **Secondary Path (Vulkan):** If the fragile ROCm drivers fail to initialize, the system automatically falls back to the **Vulkan** backend for llama.cpp. This ensures that the "Cognitive Core" is always operational, prioritizing availability over raw speed.4

### 2.3 The Application Layer: The Rust Modular Monolith

The "Ask Pete" application architecture rejects the operational complexity of microservices in favor of a **Modular Monolith**. The codebase is organized into distinct "Crates" (libraries) within a single Cargo workspace, enabling strict boundaries while deploying as a single binary.

#### 2.3.1 The Engine: Bevy ECS

At the heart of the simulation is **Bevy**, a data-driven **Entity Component System (ECS)**.

* **Entities:** Unique IDs representing Students, Trains, or Knowledge Nodes.
* **Components:** Data payloads (e.g., CargoWeight, FuelLevel, Velocity, Anxiety).
* Systems: Logic functions that iterate over components (e.g., combustion\_system, friction\_system).  
  This architecture allows the "Railway Physics" to be simulated at 60 ticks per second. Complex emergent behaviors—such as a student slowing down due to high "Social Friction" (Anxiety)—are handled dynamically by the interaction of systems rather than rigid scripting.2

#### 2.3.2 The Web Server: Axum 0.8

**Axum** handles the HTTP and WebSocket layer. Updated to version 0.8, it utilizes the new path parameter syntax (/{single}) and removes the #[async\_trait] overhead for cleaner code. Axum manages the API endpoints for the "Signal Tower" (Mentor Portal) and handles authentication via JWTs.6

#### 2.3.3 The Bridge: bevy\_defer

A critical engineering challenge is bridging the asynchronous world of the web server (Tokio/Axum) with the synchronous world of the game engine (Bevy). The architecture utilizes the **bevy\_defer** crate (v0.13+) to create an AsyncWorld resource. This allows Axum handlers to queue tasks onto the Bevy executor, ensuring thread safety without deadlocking the server. This bridge is the "Architectural Glue" that allows the web interface to query the game state in real-time.6

#### 2.3.4 The Frontend: Leptos 0.8

The user interface is built with **Leptos 0.8**, utilizing **"Islands Architecture."** This allows the application to serve static HTML for fast initial loads (SEO) while hydrating specific "Islands" of interactivity (the Train Yard canvas, the Node Garden map) with WebAssembly (Wasm). This ensures high performance even on lower-end student devices.6

### 2.4 The Data Layer: Local-First RAG

To ensure data sovereignty, the system employs a **Local-First** architecture.

* **Storage:** Student data (journals, reflections) is stored in a local **Obsidian** vault as Markdown files.
* **Ingestion:** A background service chunks this content and uses the NPU to generate embeddings.
* **Vector Database:** These vectors are stored in a local **LanceDB** or **ChromaDB** instance.
* **RAG Pipeline:** When a student queries the AI, the system retrieves relevant context from their own local history. This allows the AI to "remember" the student's journey without ever sending PII to the cloud.2

## Part III: The Cognitive Architecture — Hybrid Intelligence

The "Ask Pete" system functions as a **Hybrid Sovereign Intelligence**, fusing the privacy of local models with the reasoning power of the cloud.

### 3.1 The "Yin and Yang" of AI

The AI's role is bifurcated into two distinct functions, mirroring the Taoist concept of Yin and Yang.1

#### 3.1.1 The Yang: The Train Yard (Creative Tool)

This is the authoring domain. Instructional Designers use the "Train Yard" to build topological curriculum maps. The AI here acts as a **Co-Pilot**, suggesting narrative branches, generating asset placeholders, or brainstorming "Plot Templates" (e.g., The Hero's Journey). It empowers the creator to expand their vision.1

#### 3.1.2 The Yin: The Weigh Station (Constraining Force)

This is the objective, constraining force. The "Weigh Station" runs a background NLP process locally on the GPU. It acts as a **"Safety Lockout."**

* **Mechanism:** As the instructor builds a lesson, the Weigh Station analyzes the text to calculate its **Intrinsic Load**.
* **Constraint:** If the instructor overloads a lesson with too many heavy concepts—exceeding the calculated "Cargo Capacity" of the target student profile—the Weigh Station physically prevents the lesson from being published.
* **Outcome:** This ensures that the AI acts as a guardian of pedagogical rigor, preventing "cognitive dumping" and ensuring all content remains within the Zone of Proximal Development (ZPD).1

### 3.2 The Agentic Loop: Roo Code and Self-Coding

To support rapid evolution, the architecture incorporates a **"Self-Coding"** agent. Using **Roo Code** (a fork of Cline) connected to the local LM Studio server, the system can modify its own codebase. The agent follows a **Plan-Act-Observe-Correct** loop:

1. **Plan:** Analyze the request (e.g., "Add a fuel gauge to the UI").
2. **Act:** Write the Rust code and execute cargo build.
3. **Observe:** Read compiler errors (stderr).
4. Correct: Refactor the code until it compiles.  
   This capability allows the system to evolve its own tools, effectively turning the development environment into a semi-autonomous loop.4

### 3.3 The Privacy Moat: FERPA Compliance

The **"Hybrid Sovereign"** model is key to institutional adoption.

* **The Moat:** All sensitive student data (PII, reflections) is processed locally by the **Gemma 3** or **Llama 3** model on the student's device. This data never leaves the "Maintenance Shed."
* **The Bridge:** Only anonymized, aggregated **"Process Data"** (telemetry on coal burn rates, time-on-task, GPS nodes visited) is synchronized to the Google Cloud Platform via **Antigravity**. This satisfies **FERPA** and **IRB** requirements, allowing Purdue to use the tool for research without compromising privacy.1

## Part IV: The "Phygital" Expansion — Geolocation and LARP

The evolution of "Ask Pete" extends the Railway Metaphor into the physical world, operationalizing the **"Physical AI"** mandate via a **Geolocation/LARP (Live Action Role-Playing)** interface.

### 4.1 From Metaphor to Mechanism: Embodied Cognition

In the digital realm, "Cognitive Fuel" is a simulated variable. The Phygital expansion converts this into **Kinetic Fuel**. By forcing the student to physically traverse space to "unlock" a curriculum node, the system grounds abstract learning in physiological activity. This engages the **"Motor Coding"** systems of the brain, creating a **"Dual Coding"** effect that enhances retention and transfer.15

### 4.2 Technical Implementation: The Node Garden

The system creates a **"Digital Twin"** of the campus using open data.

* **Mapping:** The application uses the **osmpbfreader** crate to parse **OpenStreetMap (OSM)** PBF files. This allows the Bevy engine to render a "Low-Resolution" blueprint map of the campus without relying on expensive proprietary APIs (like Google Maps).15
* **Node Gardens:** Specific physical locations (e.g., the Bell Tower, the Engineering Fountain) are geofenced as "Stations."
* **Vocabulary-as-a-Mechanic (VaaM):** To solve a location-based puzzle, the student must "equip" a specific vocabulary word and apply it to the physical context. For example, unlocking the "Engineering Station" might require equipping the word "Fulcrum" and physically standing at the specific GPS coordinates.1

### 4.3 The "Low-Resolution" AR Aesthetic

To minimize extraneous cognitive load, the interface rejects hyper-realistic AR in favor of a **"Blueprint"** aesthetic. The world is rendered as a schematic map, highlighting only the relevant "Nodes" and "Tracks." This focuses the student's attention on the learning objective rather than the spectacle of the technology.15

## Part V: Strategic Implementation — The Living Laboratory

### 5.1 Purdue University as a Living Laboratory

The initiative is designed to function as a **"Living Laboratory"** for Purdue University. By capturing high-fidelity telemetry on learning processes—latency, hesitation, "coal" burn rates, GPS movement patterns—it enables **Design-Based Research (DBR)** on a massive scale. This aligns with the **Purdue-Google "Physical AI" partnership**, operationalizing the "We Thrive" pillar by creating a digital twin of the student's cognitive journey for predictive analysis (e.g., identifying students at risk of "derailment" before they fail).1

### 5.2 Departmental Roles: The Coalition

To ensure widespread adoption, "Ask Pete" is pitched as an **Organizational Tool** with specific ownership roles for diverse departments 1:

* **Computer Science (CS):** Owns **AI Security** (ensuring the "Weigh Station" cannot be jailbroken) and **UI Presentation** (optimizing the Bevy ECS visualization).
* **Learning Design & Technology (LDT):** Owns the **"Cognitive Architecture,"** defining the Intrinsic Load weights and track friction coefficients.
* **Economics:** Owns the **"Coal & Steam" Economy.** The Vernon Smith Experimental Economics Laboratory (VSEEL) models resource scarcity to prevent inflation and ensure the game economy remains balanced.
* **Creative Writing:** Owns the **"Narrative Architecture,"** creating "Plot Templates" and archetypes that allow IDs to "storyfy" their curriculum.
* **Purdue OWL:** Owns the **"Literacy Foundation,"** structuring text comprehension challenges and guiding the AI's Socratic feedback protocols.

### 5.3 Business Model: Internal Recharge and Spin-Out

To navigate university procurement hurdles and ensure long-term sustainability, the project proposes a dual-structure model 1:

1. **Internal Recharge Center:** Established within Purdue, allowing departments to "buy" access to the platform using internal funds (Journal Vouchers). This covers server costs and ensures data remains within the university's legal umbrella.
2. **Daydream Labs PBC (Spin-Out):** A Public Benefit Corporation formed by the creator. It negotiates an exclusive commercial license from the Purdue Office of Technology Commercialization (OTC). Crucially, the spin-out grants Purdue a **perpetual, royalty-free license** for internal use ("The Anchor Customer"), ensuring the university retains the research asset while the company scales commercially.

### 5.4 The "Trojan Horse" Strategy

To expand beyond academia, the project utilizes a **"Trojan Horse"** strategy for the corporate market. The platform is pitched to corporations not just for soft skills but for **"High-Technical Onboarding."** Companies use the "Train Yard" to build custom modules for their internal documentation. The "Jules" agent (a specialized persona) acts as the onsite mentor. This high-margin corporate revenue cross-subsidizes the educational mission, ensuring low costs for public schools.2

## Part VI: The Consolidation — Building the "Trinity Server"

The strategic analysis identifies **Consolidation** as the immediate bottleneck. The project currently exists as three separate prototypes. The next step is to merge them into a single, unified application: the **"Trinity Server"** (Grand Central Station).

### 6.1 The "Grand Central Station" Architecture

The "Trinity Server" will be the unified backend executable that orchestrates all traffic between the Authoring Tool, the Gameplay Engine, and the AI Constraints.

#### 6.1.1 Workspace Structure (Cargo.toml)

To formally merge the prototypes, the codebase must be reorganized into a cohesive **Cargo Workspace**. This enforces the Modular Monolith topology, separating concerns while maintaining a single build pipeline.11

**Table 1: Trinity Workspace Directory Structure**

| **Path** | **Description** | **Role in Architecture** |
| --- | --- | --- |
| trinity-root/Cargo.toml | **Workspace Root** | Defines the workspace members and shared dependencies (Tokio, Serde, Bevy). |
| apps/grand-central/ | **Server Binary** | The main executable. Runs Axum (HTTP) and Bevy (Headless ECS). |
| apps/node-garden/ | **Client Wasm** | The frontend application. Runs Leptos and Bevy (Client Mode) for the browser. |
| libs/core/ | **Shared Kernel** | Common types (CargoWeight, Coal, UserID) shared by all crates. |
| libs/domain-trainyard/ | **Authoring Domain** | Logic for the Topological Graph Editor and Lesson Constraints. |
| libs/domain-physics/ | **Simulation Domain** | Bevy Systems for the Railway Physics (combustion\_system, friction\_system). |
| libs/domain-auth/ | **Security Domain** | JWT handling, Permissions, and User Roles. |
| libs/infra-ai/ | **Intelligence Layer** | Connectors for llama.cpp (Local), Gemini (Cloud), and Vector DBs. |
| libs/infra-db/ | **Persistence Layer** | SQLx adapters for Postgres and LanceDB/ChromaDB interactions. |

#### 6.1.2 The Workspace Definition (Cargo.toml)

This file defines the federation of crates.

Ini, TOML

# trinity-root/Cargo.toml  
[workspace]  
resolver = "2"  
members = [  
 "apps/grand-central",  
 "apps/node-garden",  
 "libs/core",  
 "libs/domain-auth",  
 "libs/domain-trainyard",  
 "libs/domain-physics",  
 "libs/infra-ai",  
 "libs/infra-db",  
]  
  
[workspace.dependencies]  
# The Trinity Stack  
axum = "0.8.0"  
bevy = { version = "0.17", default-features = false } # Headless for server  
bevy\_defer = "0.13" # The Async Bridge  
leptos = "0.8"  
tokio = { version = "1.0", features = ["full"] }  
serde = { version = "1.0", features = ["derive"] }  
sqlx = { version = "0.8", features = ["runtime-tokio-rustls", "postgres"] }  
tracing = "0.1"

#### 6.1.3 The Orchestration Logic (main.rs)

The apps/grand-central/src/main.rs file acts as the "Conductor," initializing both the Game Engine and the Web Server and bridging them via bevy\_defer.

Rust

// apps/grand-central/src/main.rs  
use bevy::prelude::\*;  
use axum::Router;  
use bevy\_defer::AsyncPlugin;  
use std::net::SocketAddr;  
use tokio::net::TcpListener;  
  
// Import Domain Plugins  
use domain\_physics::PhysicsPlugin;  
use domain\_trainyard::TrainYardPlugin;  
use infra\_ai::WeighStationPlugin;  
  
#[tokio::main]  
async fn main() {  
 // 1. Initialize Tracing  
 tracing\_subscriber::fmt::init();  
  
 // 2. Initialize the Bevy ECS (The Physics Engine)  
 // We use MinimalPlugins for a headless server (no windowing).  
 let mut app = App::new();  
 app.add\_plugins(MinimalPlugins);  
   
 // 3. Install the Bridge (AsyncWorld)  
 // This enables Axum handlers to dispatch tasks to this Bevy App.  
 app.add\_plugins(AsyncPlugin::default\_settings());  
  
 // 4. Load Domain Logic (The "Ask Pete" Systems)  
 app.add\_plugins(PhysicsPlugin); // Coal/Steam Systems  
 app.add\_plugins(TrainYardPlugin); // Authoring Tools  
 app.add\_plugins(WeighStationPlugin); // AI Constraints  
  
 // 5. Initialize the Axum Router (The Signal Tower)  
 // We merge routes from different domains.  
 let router = Router::new()  
 .merge(domain\_auth::routes())  
 .merge(domain\_trainyard::api\_routes())  
 // Apply the middleware that injects the Bevy World into the request extension  
 .layer(bevy\_defer::axum\_middleware(app.world()));  
  
 // 6. Bind to Port (Cloud Run requirement)  
 let port = std::env::var("PORT").unwrap\_or\_else(|\_| "8080".to\_string());  
 let addr: SocketAddr = format!("0.0.0.0:{}", port).parse().unwrap();  
 let listener = TcpListener::bind(addr).await.unwrap();  
  
 tracing::info!("Trinity Server listening on {}", addr);  
  
 // 7. Run the Hybrid Server  
 // We must run Axum and Bevy concurrently. Bevy needs to tick to process physics.  
 // bevy\_defer provides a runner or we spawn Axum as a task.  
   
 // Option A: Spawn Axum in a Tokio task, run Bevy on main thread (Recommended for ECS priority)  
 let server\_handle = tokio::spawn(async move {  
 axum::serve(listener, router).await.unwrap();  
 });  
  
 // Run the Bevy Game Loop (The Heartbeat)  
 // This will process all physics, AI ticks, and AsyncWorld tasks from Axum.  
 app.run();  
   
 // (In a real scenario, handle graceful shutdown of both)  
 let \_ = server\_handle.await;  
}

### 6.2 Deployment Pipeline: "The Phoenix Project"

The consolidation phase is governed by **"The Phoenix Project"** remediation strategy, which outlines the build and deployment process.3

#### 6.2.1 Multi-Stage Docker Build

To ensure rapid deployment to **Google Cloud Run**, the project utilizes a multi-stage Dockerfile with **cargo-chef**. This optimizes the build time by caching the compilation of the heavy dependencies (Bevy, Axum) separate from the application code.

1. **Planner Stage:** Computes the "recipe" (dependency graph) of the workspace.
2. **Cacher Stage:** Builds all dependencies based on the recipe. This layer is cached by Docker, meaning a change in main.rs does not trigger a re-compile of the entire Bevy engine (which can take 15+ minutes).
3. **Builder Stage:** Compiles the actual grand-central binary and the node-garden Wasm client.
4. **Runtime Stage:** Copies the artifacts into a distroless "Google Cloud Run" compatible image (Debian Slim), ensuring a small footprint and high security.11

## Conclusion: The Destination is the Journey

Project "Ask Pete" (Trinity) is not merely a software application; it is a holistic response to the structural deficits of modern education. By synthesizing the "Iron Road" of Rust engineering with the "Railway Pedagogy" of Cognitive Logistics, it creates a system where the pursuit of knowledge is as engaging as a game and as rigorous as a science.

The architecture described herein—the **Compute Triad** on Strix Halo, the **Modular Monolith** in Rust, and the **Phygital Node Garden**—provides the robust foundation necessary for this vision. With the consolidation of the "Trinity Server," the project is poised to transition from a collection of prototypes into a unified, institutional-grade platform. The tracks are laid, the engine is stoked with the coal of motivation, and the signal is green. The train is ready to depart.

#### Works cited

1. The 'Ask Pete' Initiative: A Strategic Blueprint for Institutionalizing a GPS-Enabled Physical AI Learning Ecosystem at Purdue University, <https://drive.google.com/open?id=1jiB4KZhtUAAmb97Ynl41UTrh4EyinjZkUrV3U1KDFTE>
2. MMO LitRPG Builder: Architecture and Design, <https://drive.google.com/open?id=1S-p_7mWSeC2cbstjXqH_apIV6gqulZUqKJ27lmbuQMc>
3. gamelit logic thinking, <https://drive.google.com/open?id=1yGzyE5nW517Ta15VHnY9DzIf5G5svFMVjg0HsOw9R2Y>
4. Building Trinity: Autonomous AI Agent, <https://drive.google.com/open?id=1pRjPzv_PGTn03-p9Sp9-mHER97TyhfmM_FFMYvN610E>
5. Project Trinity AI Assistant Setup, <https://drive.google.com/open?id=1EmNCOTslUl02nbgOyaqm4O41lTIx1Zn3sLvTWOoIa3w>
6. Updating Project Dependency List, <https://drive.google.com/open?id=1Od9EEb-9WRPZa4ja_GHyfAwA8WFm312na4vV6rUqAzA>
7. Announcing axum 0.8.0 | Tokio - An asynchronous Rust runtime, accessed November 29, 2025, <https://tokio.rs/blog/2025-01-01-announcing-axum-0-8-0>
8. axum - crates.io: Rust Package Registry, accessed November 29, 2025, <https://crates.io/crates/axum/0.8.0>
9. bevy\_defer - Rust - Docs.rs, accessed November 29, 2025, <https://docs.rs/bevy_defer/latest/bevy_defer/>
10. mintlu8/bevy\_defer: A simple asynchronous runtime for executing async coroutines in the bevy engine. - GitHub, accessed November 29, 2025, <https://github.com/mintlu8/bevy_defer>
11. Local AI Development & Website Setup, <https://drive.google.com/open?id=1wCPqiU_aifdk_tG_cwXv2HDomOMr2pEF5lv6bD_Fsss>
12. Leptos App Development and Deployment, <https://drive.google.com/open?id=1R7q8KcAe23kbHWESMT-EYEX23ExxqrdKLODg_X2SMJY>
13. Getting Started - Leptos Book, accessed November 29, 2025, <https://book.leptos.dev/getting_started/index.html>
14. AI Education Project: Impact and Improvement, <https://drive.google.com/open?id=1uvKK6q_Zqy7w2N-DTF5vjPx1RE21Bfn9yKJCNYZbuPo>
15. Rust Geolocation LARP Web App, <https://drive.google.com/open?id=1e8HE9tSYn2FXBvVUFKnYvGjmqYAk5f_g-Y7oeC01yeM>
16. osmpbfreader - Rust - Docs.rs, accessed November 29, 2025, <https://docs.rs/osmpbfreader>
17. The best way to structure Rust web services - LogRocket Blog, accessed November 29, 2025, <https://blog.logrocket.com/best-way-structure-rust-web-services/>
18. Cargo Workspaces - The Rust Programming Language, accessed November 29, 2025, <https://doc.rust-lang.org/book/ch14-03-cargo-workspaces.html>