

Bob’s Brain Refactor Analysis

Document: 244-PP-RMAP | **Category:** PP (Project Planning) | **Type:** RMAP (Roadmap) **Date:** 2026-02-18 | **Author:** Intent Solutions **Status:** Decision Analysis

Purpose

Evaluate language/framework options for refactoring Bob’s Brain, the enterprise orchestrator currently implemented in Python with Google ADK on Vertex AI Agent Engine.

Current State

Attribute	Value
Repository	intent-solutions-io/iam-bobs-brain (org repo, not personal)
Language	Python
Framework	Google ADK (Agent Development Kit)
Runtime	Vertex AI Agent Engine
Architecture	Risk tiers R0-R4, policy gates, evidence bundles, Mission Spec v1
Status	Production-grade

Key Capabilities

- **Risk Tier Governance:** R0 (read-only) through R4 (financial, requires human + IRSB receipt)
- **Policy Gates:** Enforce approval requirements before operations execute
- **Evidence Bundles:** Collect and preserve audit evidence for every decision
- **Mission Spec v1:** Structured mission definitions with success criteria

Refactor Options

Option 1: TypeScript

Factor	Assessment
Rationale	Matches GWI, Automaton, MCP servers. Unified stack across ecosystem
ADK Support	Google ADK has TypeScript SDK support
Ecosystem Fit	Excellent — aligns with 3 major projects
Effort	Medium (2-4 weeks)
Risk	Low — well-understood language, team expertise exists

Pros: - Single language across GWI, Automaton, and Bob - Shared tooling (Vitest, Turbo, ESLint)
- TypeScript type system catches errors at compile time - npm ecosystem for MCP, A2A protocol libraries

Cons: - ADK Python SDK is more mature than TypeScript - Some Vertex AI features are Python-first - Existing Python codebase is production-grade — rewrite introduces regression risk

Option 2: Go

Factor	Assessment
Rationale	Fast, single binary, great for infrastructure agents
ADK Support	No native Google ADK support — would use REST API
Ecosystem Fit	Cortex (“AI Layer for Linux”) aligns with Go’s systems programming strength
Effort	High (4-8 weeks)
Risk	Medium — new language for the team, no ADK SDK

Pros: - Single binary deployment, no runtime dependencies - Excellent concurrency model (goroutines, channels) - Strong fit for Cortex vision (OS-level agent) - Fast startup, low memory footprint

Cons: - No Google ADK SDK for Go — must use REST API directly - Team would need to learn Go patterns - Less mature AI/ML ecosystem compared to Python/TypeScript - Vertex AI client libraries exist but are less documented

Option 3: Rust

Factor	Assessment
Rationale	Performance, safety, WASM compilation, future-proof
ADK Support	No native support — REST API only
Ecosystem Fit	Niche — only relevant if building performance-critical infrastructure
Effort	Very High (8-16 weeks)
Risk	High — steep learning curve, small ecosystem for AI agents

Pros: - Memory safety guarantees (no GC pauses) - WASM compilation for portable execution - Best performance characteristics - Growing web/API ecosystem (Axum, Actix)

Cons: - Steepest learning curve of all options - Smallest AI/ML ecosystem - Overkill for orchestration workloads - Slowest development velocity

Option 4: Stay Python

Factor	Assessment
Rationale	ADK Python SDK is most mature. Vertex AI native. LangChain ecosystem
ADK Support	First-class — Python SDK is the primary SDK

Factor	Assessment
Ecosystem Fit	Isolated from TypeScript projects but strongest AI/ML ecosystem
Effort	Zero
Risk	None — current system is production-grade

Pros: - Zero migration effort - Most mature ADK SDK - Richest AI/ML library ecosystem - Vertex AI client libraries are Python-first - Current codebase is production-grade and tested

Cons: - Language fragmentation with GWI/Automaton (TypeScript) - Separate tooling and CI/CD pipelines - No shared type definitions across projects - Performance overhead for compute-intensive orchestration

Decision Framework

If the goal is unifying with Automaton/GWI → TypeScript

TypeScript creates a single language across the three most strategic projects. Shared types, shared tooling, shared CI/CD. The ADK TypeScript SDK is maturing and sufficient for orchestration workloads.

If the goal is standalone infra agent (Cortex vision) → Go

Go excels at systems programming and infrastructure agents. If Bob evolves into a system-level orchestrator (aligned with Cortex), Go is the natural choice. But this diverges from the agent ecosystem.

If ADK maturity matters most → Stay Python

Python gives the richest ADK experience with zero migration risk. If Bob's Brain is working well in production, there is no compelling reason to rewrite. "If it ain't broke, don't fix it."

Recommendation

Scenario	Choice	Reasoning
Default	Stay Python	Production system works. No urgent need to rewrite
Ecosystem unification	TypeScript	Aligns with GWI + Automaton. Justified if deep integration proceeds
Cortex vision	Go	Only if Bob becomes a system-level agent
Never	Rust	Overkill for orchestration. Wrong tradeoff curve

Recommended Path

1. **Short term (0-3 months):** Stay Python. Focus engineering effort on Automaton integration, not Bob rewrite
2. **Medium term (3-6 months):** If Option C integration proves valuable, evaluate TypeScript port as Phase 5
3. **Long term (6-12 months):** If Cortex materializes, consider Go for the system-level component only

The refactor decision should follow the integration strategy, not lead it. Build the integrations first (Phases 1-4), then decide if language unification adds enough value to justify the migration cost.

Document follows 6767 Filing Standard v4.2