

**TRAVIA AI ARCHITECTURE - DETAILED EXPLANATION**

**1. DATA LAYER**

**Supabase Database**

* **What it is**: Your existing PostgreSQL database that stores all user data, posts, trip history, and preferences
* **Why important**: Central source of truth for all application data
* **Role in AI**: Provides training data and real-time user context

**Vector Database (Pinecone/Weaviate)**

* **What it is**: Specialized database for storing high-dimensional vectors (embeddings)
* **Why needed**: Fast similarity search for recommendations
* **Technologies**: Pinecone (cloud), Weaviate (open-source), or Chroma
* **Stores**: Business embeddings, user preference vectors, content representations

**Feature Store (Redis/DynamoDB)**

* **What it is**: Cache for frequently accessed, pre-computed features
* **Why important**: Real-time serving with low latency
* **Technologies**: Redis for hot data, DynamoDB for warm data
* **Contains**: User behavior patterns, popular destinations, real-time counters

**2. TRIP PLANNING SYSTEM**

**Content-Based Filtering for Cold Start**

**What is Cold Start Problem?**

* **Problem**: New users have no history, so traditional collaborative filtering fails
* **Solution**: Use content features (business attributes, location, categories) to make recommendations

**Feature Extraction Components:**

1. **TF-IDF (Term Frequency-Inverse Document Frequency)**
   * **What it does**: Converts text descriptions into numerical vectors
   * **How**: Measures importance of words in business descriptions
   * **Technology**: scikit-learn's TfidfVectorizer
2. **Word2Vec/Sentence Embeddings**
   * **What it does**: Creates semantic understanding of text
   * **How**: Maps similar words/sentences close in vector space
   * **Technology**: Google Word2Vec, OpenAI embeddings, or Sentence-BERT
3. **Categorical Features**
   * **What**: Business types (restaurant, hotel, attraction)
   * **Processing**: One-hot encoding or embedding layers
   * **Example**: "Italian Restaurant" → [0,1,0,0,0] vector

**Similarity Engine:**

* **Cosine Similarity**: Measures angle between vectors (good for text)
* **Euclidean Distance**: Measures straight-line distance between points
* **Jaccard Index**: Measures overlap between sets (good for categories)

**Reinforcement Learning Enhancement**

**Key RL Concepts:**

1. **State**: Current situation
   * User's current trip progress
   * Visited places so far
   * Time remaining
   * Budget spent
   * Current location
2. **Action**: What the AI can do
   * Suggest a specific business/attraction
   * Modify route order
   * Adjust time allocation
   * Change transportation mode
3. **Reward**: Feedback signal
   * +1 for user acceptance
   * +5 for 5-star rating
   * -1 for rejection
   * +3 for trip completion
   * Bonus for budget efficiency
4. **Q-Learning Agent**
   * **What**: Learns value of actions in each state
   * **How**: Updates Q-table: Q(state, action) = reward + γ \* max(future\_rewards)
   * **Technology**: Stable-Baselines3, TensorFlow Agents, or custom implementation

**Trip Planning Pipeline Components:**

1. **Route Optimizer**
   * **Purpose**: Find optimal travel routes
   * **Technology**: Google Maps API, OpenRouteService
   * **Algorithm**: Traveling Salesman Problem (TSP) solvers
2. **Constraint Solver**
   * **What**: Ensures recommendations meet user constraints
   * **Constraints**: Budget limits, time windows, accessibility needs
   * **Technology**: OR-Tools (Google's optimization library)
3. **Diversity Engine**
   * **Purpose**: Avoid monotonous recommendations
   * **Methods**: Category balancing, novelty injection
   * **Algorithm**: Diversified ranking (MMR - Maximal Marginal Relevance)

**3. SOCIAL MEDIA RECOMMENDATIONS (MONOLITH)**

**Monolith Approach Explained**

**Traditional vs. Monolith:**

* **Traditional**: Separate training and serving (batch updates)
* **Monolith**: Continuous online training with real-time updates

**Key Innovations:**

1. **Collisionless Embedding Table**
   * **Problem**: Hash collisions reduce model quality
   * **Solution**: Cuckoo Hashing ensures no collisions
   * **Benefit**: Each user/post gets unique representation
2. **Real-time Online Training**
   * **How**: Model updates every few minutes from user interactions
   * **Technology**: Apache Kafka for streaming, Apache Flink for processing
   * **Benefit**: Captures concept drift (changing user preferences)

**Cuckoo Hashing Explained:**

* **Concept**: Uses two hash tables with different hash functions
* **Process**: If slot is occupied, "kicks out" existing item to other table
* **Advantage**: Guaranteed O(1) lookup time, no collisions
* **Memory Management**: Frequency filtering removes infrequent items

**Multi-Tower Architecture:**

1. **User Tower**
   * **Input**: User demographics, travel history, preferences
   * **Output**: User embedding vector
   * **Network**: Deep neural network (3-4 layers)
2. **Content Tower**
   * **Input**: Post features (images, text, location, hashtags)
   * **Output**: Content embedding vector
   * **Processing**: CNN for images, BERT for text
3. **Context Tower**
   * **Input**: Situational factors (time, device, weather)
   * **Output**: Context embedding vector
   * **Features**: Time of day, season, device type
4. **Interaction Tower**
   * **Input**: Historical interaction patterns
   * **Output**: Interaction preference embedding
   * **Features**: Like/save/share patterns, dwell time

**Deep FM (Factorization Machine + Deep Neural Network):**

* **FM Component**: Captures feature interactions (like collaborative filtering)
* **Deep Component**: Learns complex patterns
* **Fusion**: Combines both for final prediction

**4. ML INFRASTRUCTURE**

**Model Serving**

* **FastAPI/Flask**: Python web frameworks for API endpoints
* **TensorFlow Serving**: Production-ready model serving system
* **Load Balancer**: Distributes requests across multiple model instances
* **Auto-scaling**: Automatically adds/removes servers based on load

**MLOps Pipeline**

* **MLflow**: Experiment tracking and model versioning
* **Kubeflow**: Machine learning workflows on Kubernetes
* **A/B Testing**: Compare model versions with real users
* **Model Monitoring**: Detect performance degradation

**Container Orchestration**

* **Kubernetes**: Container orchestration platform
* **Docker**: Containerization technology
* **Helm Charts**: Package manager for Kubernetes
* **Service Mesh**: Network communication between services

**INTEGRATION STRATEGY**

**Cold Start Solution Steps:**

1. **New User Onboarding**: Questionnaire about preferences
2. **Content-Based Initial**: Use business features for first recommendations
3. **Quick Learning**: RL agent adapts after 2-3 interactions
4. **Cross-Domain Transfer**: Use social media activity to infer travel preferences

**Technology Stack by Component:**

**Trip Planning:**

* **Backend**: Python, FastAPI
* **ML Libraries**: scikit-learn, TensorFlow/PyTorch
* **Optimization**: OR-Tools, NetworkX
* **Maps**: Google Maps API, Mapbox
* **RL**: Stable-Baselines3, Ray RLlib

**Social Recommendations:**

* **Streaming**: Apache Kafka, Apache Flink
* **ML Framework**: TensorFlow/PyTorch
* **Serving**: TensorFlow Serving, TorchServe
* **Caching**: Redis, Memcached
* **Database**: PostgreSQL (Supabase), Cassandra

**Infrastructure:**

* **Cloud**: AWS/GCP/Azure
* **Containers**: Docker, Kubernetes
* **Monitoring**: Prometheus, Grafana
* **CI/CD**: GitHub Actions, Jenkins
* **GPU**: NVIDIA T4/V100 for training

**Deployment Strategy:**

1. **Development**: Local Docker containers
2. **Staging**: Kubernetes cluster with subset of data
3. **Production**: Multi-region deployment with auto-scaling
4. **Monitoring**: Real-time performance dashboards
5. **Rollback**: Blue-green deployment for safe updates

This architecture provides a robust, scalable AI system that can handle both cold start problems in trip planning and real-time learning for social media recommendations, following industry best practices from the Monolith paper.