(1) pick the best high-accuracy, production-ready approach, (2) explain why it's the best choice vs the other options, (3) show fallbacks (if vectors are a poor fit), (4) propose concrete tech choices and index/tuning guidance, and (5) give an actionable step-by-step roadmap + test plan you can hand your manager.

I used your two PDFs as the baseline for requirements and constraints. I also checked modern, practical industry patterns for large-scale ANN, two-stage retrieval, and vector DB tradeoffs to make sure the recommendations match current best practice.

TL;DR — Best approach (one-line)

Use a **hybrid two-stage pipeline**:

(1) Candidate generation: two-tower or

collaborative embedding model → ANN search (HNSW/IVF+PQ) in a vector DB (prototype with FAISS or Qdrant; prod with Milvus or Pinecone).
(2) Re-ranking: fast learned ranker that combines vector score + business signals (propensity, freshness, budget fit, suppression rules). Use offline evaluation + online A/B tests. This gives highest accuracy + production practicality.

Why this (short reasoning)

- **Vectors are the right fit** for user ↔ trip matching because both customers and trips can be represented as embeddings (behavioral/collaborative + content features) and nearest-neighbour returns semantically similar trips. Your PDFs already assume vectors; that's standard.
- **Two-stage is industry standard**

(candidate generation by ANN, then re-ranking) for balancing recall, latency, and business logic — used by Netflix/Spotify/Pinterest. It's how you get high-quality recommendations at scale.

- **Vector DB + re-rank gives accuracy + control**: vector search finds semantically good matches; re-rank adds business constraints (consent, budget, freshness), suppression and personalization.

Full recommended architecture (high level)

- **Data & Feature Store**
- Raw tables per your brief: customer, flight_order, offer_catalog. Compute RFM, route affinity, channel preferences, household proxies, seasonality into a feature store (Redis/Feast/Postgres). Use pseudonymous

keys and respect consent/suppression rules.

- **Embedding models**
- **Two-tower model** (user tower, journey/tour tower) trained on interaction history (bookings, clicks, opens). Or hybrid: collaborative (interaction based) + content (offer tags, destination text embeddings). Two-tower lets you precompute item (journey) embeddings and user embeddings independently for fast retrieval.
- **Vector Index / Candidate Gen**
- Store journey embeddings in a vector DB (Milvus/Pinecone/Qdrant) or FAISS for prototype. Use ANN index (HNSW for high recall or IVF+PQ for memory efficiency at huge scale). Pre-filter by metadata (region, consent, channel) to reduce search space (two-stage

hybrid).

- **Re-ranking**
- Lightweight ranker that takes candidate list + features (vector score, propensity score, recency, budget fit, channel preference, suppression) → final sorted list. Can be a gradient boosted tree (XGBoost/LightGBM) for fast inference, or a small NN for better non-linear combinations.
- **Serving & Orchestration**
- Online: On user session, fetch user embedding (or compute from recent events), query vector DB for top-N, re-rank, apply suppression/cadence rules, return top K for UI/email.
- Offline: nightly batch precompute top-K for all users to drive email campaigns (makes sending at scale cheap).
- **Monitoring & Feedback**
- Track CTR, conversion, precision@K,

recall@K, NDCG, revenue uplift, opt-out rate. Use these metrics for model retraining cadence and index tuning.

Concrete tech choices (prototype → prod)

- **Prototype / dev**: Python, PyTorch/ TensorFlow, FAISS (local), Postgres for metadata, Redis for fast lookups, Airflow/Prefect for pipelines. (FAISS is ideal to experiment locally and validate indexes.)
- **Production (managed)**: Milvus or Pinecone (managed vector DB), or Qdrant if you want OSS + hosted. Use Kafka for event streaming (interaction logs), Airflow for ETL, Redis for feature cache, Triton or a lightweight model server for ranker inference. Use Kubernetes for orchestration.

Why Milvus/Pinecone? they handle

sharding/replication, scale to 10^8 vectors, and let you choose index types (HNSW/IVF/PQ) appropriate to your scale. If you want fully self-managed and GPU acceleration, FAISS + custom sharding is possible but more operational work.

Which index to pick (practical rules)

- **< 1M journeys**: HNSW or IVF variants on FAISS — high recall, low-latency; prototyping with FAISS is fine.
- **1M–10M**: IVF+HNSW or IVF65536_HNSW32 as suggested by FAISS guidelines for this range. Tune K and PQ quantization.
- **100M+**: Use distributed vector DB (Milvus/Pinecone) with sharding and PQ compression to reduce RAM. HNSW gives best recall but needs memory; IVF+PQ is more memory

efficient.

(Tip: pre-filter by geography/season to lower the index search set by 10–100x — this is in your Journey doc as “Two-stage hybrid”.)

What if *vectors* turn out to be a poor fit?

Vectors are usually strong for semantic matching. But if:

- interactions are extremely sparse, or
- you have strong, tabular signals (budget bands, fixed routes with few variations), then:
- Use **feature + propensity models** (XGBoost / logistic regression) as fallback for direct propensity score (propensity to purchase a given offer). Or combine both: use a GBDT ranker that incorporates vector score as a feature. This hybrid often outperforms

pure vector-only pipelines on sparse/ low-data segments.

Evaluation & testing plan (offline → online)

Offline experiments

- **Split**: time-based train / validation (simulate real future).
- **Metrics**: Precision@K, Recall@K, NDCG@K, MAP, AUC for propensity. Also business KPIs (CTR, conversion, revenue per send).
- **Ablation tests**: vector-only vs vector+content vs propensity-only vs hybrid two-tower + re-rank.
- **Index recall tests**: measure ANN recall (how often true positive in exact top-K appears in ANN candidates) for different index parameters.

Online

- **Small-scale canary**: serve model to 1–

5% audience; measure CTR, conversion, optouts.
- **Full A/B**: control (current one-size outreach) vs treatment (new recommender); measure statistically significant improvements on core KPIs.
- **Safety**: monitor opt-outs, spam complaints, and enforce consent suppression per B2C doc.

Index & production tuning checklist (practical)

- Start with FAISS HNSW or IVF for local testing. Use FAISS wiki guidelines for K and training set sizes.
- Measure ANN recall vs latency for different ef_search / nprobe settings (HNSW/IVF param names vary).
- Use PQ compression only after validating that accuracy drop is acceptable — PQ saves memory but

reduces exactness.
- Shard by geography/market to reduce search cost and enable locality.
- Re-index periodically (or use incremental indexing) when embedding model changes. Re-training/trimming cadence depends on drift.

Concrete step-by-step roadmap you can follow this week → 12 weeks

Sprint 0 — Align & infra (1 week)

- Confirm SLAs: latency target (<100ms online retrieval), batch window, campaign cadence, privacy/consent rules. (Use B2C brief for constraints.)

Sprint 1 — Data & feature store (1 week) 2. Load sample data, implement ETL to canonical tables; compute RFM, route affinity, channel flags into a feature store

(Postgres/Feast).
Sprint 2 — Prototype embeddings (2 weeks) 3. Build small two-tower model or item2vec: train on interaction data (PyTorch). Precompute journey embeddings and sample user embeddings. Validate nearest neighbors qualitatively.
Sprint 3 — Prototype ANN & retrieval (1 week) 4. Prototype retrieval in FAISS or Qdrant locally (HNSW). Evaluate recall/ latency and do pre-filtering by metadata.
Sprint 4 — Re-rank & offline eval (2 weeks) 5. Train a re-ranker (XGBoost) using vector score + features. Run offline evaluation (precision@K, NDCG) and compare alternatives (vector-only, propensity only, hybrid).
Sprint 5 — Productionize & scale (2–4 weeks) 6. Deploy vector DB (Milvus/ Pinecone). Add event streaming for

realtime updates, build API for online serving, implement suppression/cadence engine.
Sprint 6 — Test & launch (2 weeks) 7. Canary → A/B test → roll out to campaigns. Monitor business KPIs and iterate.

Example decision tradeoffs you can present to your manager

- **Highest recall, best quality**: Two-tower + HNSW (more RAM). Good if you want best recommendations and have RAM/ infra budget.
- **Budget constrained, very large corpus**: IVF + PQ on sharded cluster (Milvus) — memory efficient, slightly lower accuracy.
- **Simplicity / fastest time to demo**:

FAISS local + offline precompute + simple rule-based matching (from B2C doc) for campaigns.

Concrete “why we chose this” summary (talking points for your manager)

- **Accuracy**: Vector embeddings capture latent preferences and two-tower enables learning cross-signal similarity — higher relevance than naive rule matching.
- **Scale**: ANN indexes allow matching millions×millions efficiently; we can shard and use managed vector DBs to scale to production.
- **Business control**: Re-ranking + filtering layer enforces budget tiers, contact cadence and consent — necessary per B2C brief.

Short checklist you can paste into the doc you'll submit

- [ ] Build feature store: RFM, route affinity, channel preferences.
- [ ] Prototype two-tower embeddings (user & journey) and generate sample matches.
- [ ] Prototype ANN candidate gen (FAISS HNSW), measure recall/latency.
- [ ] Train re-ranker using vector score + features; evaluate offline metrics.
- [ ] Deploy vector DB (Milvus/Pinecone) for prod, implement online pipeline + suppression rules.
- [ ] Run canary + A/B test; measure CTR, conversion, opt-out.

If you want, I can *right now*:

- produce a 1-page architecture diagram

and a one-page design doc to hand to your manager, or

- generate the prototype code snippet (a small two-tower training loop + FAISS indexing query example) to start experimentation.

Which of those would help you most next?