ChatGPT

AI-Assisted Ticket Triage and Response System (Capstone Project Plan)

Project Overview and Architecture

We propose a **Retrieval-Augmented Generation (RAG)** pipeline that ingests new support tickets, classifies and routes them, and generates draft responses using LLMs enhanced by a lightweight knowledge base. Incoming tickets (e.g. from email, web form or Intercom) are sent via Zapier or webhooks into our workflow. The first step is **AI-driven classification/routing**: a prompt-based or embedding-based model (e.g. OpenAI GPT-3.5 turbo) summarizes the ticket text and predicts its category (product area, issue type) and priority/impact. For example, Retool's support triage uses a webhook from Intercom and runs each ticket's text through an AI action to "summarise the ticket and assign the impact" (low/med/high/critical) and product category [1]. Our system will do similarly, either by prompting GPT with examples or by embedding the ticket and comparing it to example-category vectors in Pinecone.

Next, **semantic search and RAG**: since no KB exists yet, we will create a small initial knowledge base (documents, FAQ entries or product manuals relevant to expected queries). These docs are **chunked and embedded** (e.g. via OpenAI embeddings or Pinecone Inference) and stored in a Pinecone vector index. As AWS describes, pre-processing "converts the data into text and splits it into manageable pieces… then written to a vector index" [2]. At runtime, a user's question is embedded and nearest-neighbor search retrieves the most relevant passages. The question plus retrieved snippets are then fed to the LLM to **generate a response**. In RAG, "the query vector is used to find chunks that are semantically similar," and "the prompt is augmented with additional context from the chunks" before answering [3]. This yields answers grounded in up-to-date internal content. Pinecone notes that RAG (combining an LLM with an external vector DB) produces "more accurate, up-to-date responses" and can halve hallucinations [4] [3].

Overall architecture: ticket → Zapier trigger → classification (OpenAI) → route assignment → (if automated) response generation via RAG. A management UI (built in Retool) shows tickets, AI summaries, suggested replies, and lets an agent approve or edit before sending. The flow can Slack/PagerDuty-alert on "critical" tickets. This fully-hosted pipeline avoids building custom servers: we rely on Zapier for orchestration, Pinecone for vector search, OpenAI for NLP, and Retool for the frontend.

Implementation Timeline (8 Weeks)

Week 1 – Planning & Setup: Project kickoff; define categories/priorities and success metrics; allocate team roles (e.g. 1 ML engineer, 2 backend/integration developers, 1 UI dev, 1 data/test lead). Sign up for necessary accounts: OpenAI, Pinecone, Zapier, Retool, etc. Sketch the data flow diagram.
Week 2 – Basic Ticket Flow & Tools: Prototype ticket ingestion: e.g. use Zapier or n8n to capture form/email data into a stub (Airtable or Retool database). Build a simple Zap/flow that triggers on new ticket and passes text to OpenAI via API (or Zapier's GPT action). Experiment with prompt(s) to get a basic summary and category output. Meanwhile, study Pinecone's free tier (P1 pod) and test creating a vector index.

Week 3 – Classification & Routing: Finalize ticket categories and priorities. Using GPT-3.5-turbo, craft prompts or fine-tune classification (if needed) so that the model can label ticket type and urgency. Compare with a simple embedding approach: create small "category documents" and use Pinecone's vector similarity (Retool does this by storing example vectors) [5]. Implement logic (via Zap or custom function) that takes the model output and assigns the ticket to a team/queue in the UI.
Week 4 – Knowledge Base Creation: With no historical KB, generate starter knowledge items. **Synthetic data approach:** Use tools (e.g. OpenTicketAI synthetic ticket generator [6] or manually seed with likely issues) to write sample FAQs or support articles. Chunk these docs (e.g. 200-word passages) and index them in Pinecone. Verify vector search returns relevant passages for test queries.
Week 5 – RAG Integration: Integrate semantic search into response flow. When a ticket is to be answered (or previewed), retrieve top-K relevant docs from Pinecone based on the ticket text embedding. Construct an LLM prompt that includes the user's question + relevant document excerpts. For example, "According to our knowledge base: [doc1 snippet] [doc2 snippet] … Answer the question." Generate a draft response via OpenAI. Ensure the answer references or is consistent with the retrieved content.
Week 6 – UI & Automation: Build the support dashboard in Retool (or similar). Show incoming tickets, AI-suggested category/priority, and AI-generated draft reply. Enable a human to edit and send. Use Zapier (or Retool Agents) to wire this: e.g., a Zap updates the Retool database when an answer is approved. Include notifications: e.g. send Slack if a "critical" ticket appears (as in the Retool example [7]).
Week 7 – Synthetic Testing & Refinement: Generate or simulate ~1000 tickets (see below) and push them through the pipeline. Collect metrics: classification accuracy, response relevance. Adjust prompts, thresholds, and prompt engineering. Incorporate human-in-the-loop feedback: have a teammate review AI outputs and label errors. Use this to tweak the system (e.g. add negative examples to prompts). Ensure backups: set up Pinecone backups and maintain prompt logs.
Week 8 – Evaluation & Delivery: Measure system performance against metrics (below). Prepare final documentation and demo. Compare average response time with AI vs. typical human time. Finalize cost estimates and ensure system is stable. A final sprint for polish, UI improvements, and a rehearsed demo.

Tools, Services and Cost Estimates

We choose **hosted SaaS** to minimize ops overhead. Estimated costs assume ~1,000 new tickets/week.

- **OpenAI (GPT-3.5 Turbo API):** Used for classification, summarization, and response generation. Pricing is around **$0.003 per 1K input tokens** and **$0.006 per 1K output tokens** [8]. If each ticket interaction uses ~300 input tokens and ~300 output tokens, that's 600 tokens (0.6K) per ticket ≈ $0.0018 per ticket. At 1,000/week, that's ~$1.8/week (~$7.2/mo). Even with buffers and occasional GPT-4 tests, expect under **$20–50/month** for this volume. (GPT-4 is much higher, so we stick to 3.5 for affordability.)
- **Pinecone Vector DB:** For semantic search. Pinecone's **Free tier** includes a single P1 pod (enough for ~1M vectors) at no cost [9]. Our scale is tiny (tens of thousands of vectors max), so the free plan suffices. If needed, writes beyond free are $4 per million and reads $16 per million [10]. With ~1K searches/week (~52K/year), the extra cost is negligible (<$1). Thus Pinecone cost ≈ **$0/mo** for our prototype.
- **Zapier:** To glue triggers and APIs. Zapier's free plan (100 tasks/mo) is too small. The Professional plan ($19.99/mo) covers 750 tasks [11], still below our ~4K tasks/mo. We recommend the **Team plan** (starting $69/mo) which includes more tasks and multi-user access [12]. So budget ~**$70–100/month** for Zapier to handle ticket ingestion, calling OpenAI, and updating the database. (Alternatives: n8n.cloud or Make.com could be evaluated for lower cost if needed.)

- **Retool (or similar low-code UI):** For the support dashboard. Retool's **Team plan** is **$10 per user/month** [13]. With 5 team members, that's ~$50/mo. This allows multiple devs to build apps. (A Business plan is $50/user, which we likely don't need.) If Retool is not available, alternatives like AirTable Interfaces or a simple React/Vue app on Heroku could be used, but Retool speeds up dev.
- **Hosting/DB:** Aside from the above, we can use free-tier or low-cost services. For example, a small Node/Next.js app (if needed) on Vercel/Heroku can handle any custom logic (~free to $7/mo). We can store ticket states in Airtable (free up to ~1200 records) or Zapier Tables (included). No significant cloud infra costs are expected beyond these SaaS fees.
- **Knowledge Base content:** If we host static KB pages (e.g. on GitHub Pages or Notion), cost is negligible. If we use Pinecone Assistant or Hugging Face Inference API for embeddings, budget a small amount (Pinecone's Inference for embedding, $0.16 per million tokens after free [14] — again tiny at our scale).

Summary of monthly costs: roughly OpenAI $20, Pinecone $0, Zapier $70, Retool $50 = ~$140/month in early testing. This is easily under many enterprise AI tools, and can scale up with usage.

Synthetic Data and Knowledge Base Generation

With no existing tickets or KB, we must create our own data for testing and initial training. Strategies:

- **Synthetic Tickets:** Use GPT to create realistic support tickets. For example, the **OpenTicketAI** tool is an open-source Python generator that can produce thousands of multilingual ticket records (subject, body, type, queue, priority, tags, and even a first AI reply) [6]. We can adapt its prompts or write our own: e.g., prompt GPT-4 to "Write a support email from a customer about a login failure" etc. Keep diversity: include billing, technical, account issues, etc. Also download public datasets like the Kaggle *Customer Support on Twitter* or *Multilingual Support Tickets* for variety. These tickets will be used to tune prompts and test routing accuracy.

- **Knowledge Base Content:** Brainstorm a few core FAQ pages or product guides that cover anticipated issues. We could manually write 10–20 short docs, or ask GPT to draft them (then edit for accuracy). Example: a "Getting Started" guide, password reset help, pricing info, etc. Each document should be a few paragraphs. We will then chunk these (e.g. 150–200 word pieces) and embed them. This simulates having product docs to pull answers from.

- **Iteration and Augmentation:** As the system runs, any real or test tickets with good agent answers can be added to the KB. We should also consider creating a "feedback loop" (see enhancements) where user corrections or new agent answers get added as new documents. Synthetic generation can continue: e.g., run GPT on existing tickets to create variant questions ("paraphrase node" as in [4†L139-L147]).

Evaluation Metrics

To track success, we will measure both **efficiency gains** and **response quality**:

- **Classification/Routing Accuracy:** Percentage of tickets correctly categorized (team/queue) by the AI vs. ground truth. We can compute precision/recall per category on a test set of tickets. A high accuracy (e.g. >85%) ensures few mis-routes. The Retool example noted that even a "basic prompt

does a fantastic job of summarising and correctly categorising issues" [15], but we will quantitatively measure ours.

- **Response Quality (Accuracy Rate):** For AI-generated replies, we use a small blind test: have human evaluators rate if the response "correctly addresses the customer's issue" on a sample of tickets. FeedbackRobot defines an "accuracy rate" as the percent of AI responses that correctly solve the query [16]. We aim for at least ~80–90% accuracy on routine queries. We can also include a CSAT survey (users rate the AI answer) – AI-driven CSAT is a key metric [17].

- **Time Savings:** Measure average time to first draft and final resolution. For example, track the time an agent spends composing a reply with AI assistance vs. without. Metrics like **Average Resolution Time** (from ticket arrival to closure) can be compared to historical norms [18]. If the AI handles simple tickets end-to-end, measure the proportion of tickets that needed no human reply ("contained" by AI) vs. baseline. Even reducing average handle time by a minute or two per ticket is valuable.

- **Automation/Containment Rate:** Fraction of tickets fully handled by AI (no escalation). FeedbackRobot notes "automation rate" (AI-handled interactions) and "containment rate" (no human needed) as important KPIs [19] [20]. We should track what % of incoming tickets receive an AI draft that needs no human edits.

- **User Satisfaction and Error Rates:** Track help-desk CSAT for bot responses, number of user clarifications, and misclassification incidents. Monitor any serious failures (offensive/hallucinatory content) and refine prompts.

Regular dashboards (in Retool or via Zapier Tables) will log these metrics. We will iterate until accuracy and quality meet a practical bar (e.g. 90%+ correct classification and usable replies most of the time).

Optional Enhancements

Beyond the MVP, we suggest these improvements:

- **Multilingual Support:** Since tools like GPT-4o and OpenTicketAI support multiple languages [6], we can extend to non-English tickets. Pinecone embeds (or OpenAI's multilingual models) can handle queries in DE/FR/ES/… if our knowledge base is translated or written in those languages.

- **Feedback Loop & Fine-Tuning:** Implement a loop where agent edits or reassignments are logged as new training examples. Over time, we could fine-tune a small classification model on the synthetic+feedback data. Retool's vector system (like example guidelines) suggests using feedback to improve prompts [21]. A custom "Assistant" via OpenAI's new tools (Assistant API) could also be fine-tuned on our ticket/response corpus for better domain fit.

- **Escalation Prediction:** Build a model to flag tickets that likely need human escalation (e.g. high urgency or low AI confidence). This could use sentiment analysis or GPT to predict when the AI is unsure. Early alerts ensure such cases get prompt human review.

- **Analytics Dashboard:** Use Retool's dashboards or Zapier tables to monitor ticket volumes, categorize trends, and alert if, say, a new error type emerges.

- **Channel Integration:** Extend beyond email – e.g. SMS, Slack, or social media queries – by plugging additional Zapier triggers.

Each of these can be slotted into the 8-week timeline if time allows, or as future work beyond the capstone deadline.

By using hosted APIs (OpenAI, Pinecone) and low-code tools (Zapier, Retool), this plan minimizes custom infra while delivering a production-like AI support system. The outlined schedule and tools allow a 5-person team to work in parallel (e.g. two on data/LLMs, two on integration/UI, one on testing/simulations) and achieve a working prototype in two months.

Sources: We base design choices on industry examples (e.g. Retool's AI ticket triage flow [1]) and vendor docs: Pinecone on RAG [4], AWS on RAG pipelines [2] [3], synthetic data generators [6], and published pricing for OpenAI [8], Zapier [11] [12], Retool [13], Pinecone [9] [10], plus CRM support metrics [16] [18]. All citations are given in-text.

---

[1] [5] [7] [15] [21] How Retool Triages Support Tickets With AI, Allowing Us To Immediately Identify Critical Issues And Incidents - Community Show & Tell - Retool Forum
https://community.retool.com/t/how-retool-triages-support-tickets-with-ai-allowing-us-to-immediately-identify-critical-issues-and-incidents/34758

[2] [3] How Amazon Bedrock knowledge bases work - Amazon Bedrock
https://docs.aws.amazon.com/bedrock/latest/userguide/kb-how-it-works.html

[4] RAG | Pinecone
https://www.pinecone.io/solutions/rag/

[6] Synthetic Data Generation for Support Tickets | Open Ticket AI
https://open-ticket-ai.com/en/products/synthetic-data/synthetic-data-generation

[8] Pricing | OpenAI API
https://platform.openai.com/docs/pricing

[9] Pinecone 2.0 is Available and Free | Pinecone
https://www.pinecone.io/blog/v2-pricing/

[10] [14] Pricing | Pinecone
https://www.pinecone.io/pricing/

[11] [12] Plans & Pricing | Zapier
https://zapier.com/pricing

[13] Retool | Pricing
https://retool.com/pricing

[16] [17] [18] [19] [20] Metrics to Evaluate AI Customer Service in the US | FeedbackRobot
https://www.feedbackrobot.com/blog/metrics-ai-customer-service-usa