2024

GenAI for Customer Support

“The Yoky Project”

Solution Brief

InterSystems
Creative data technology

CONTENTS

Executive Summary

InterSystems has developed the Yoky Support Assistant, an internal generative AI-powered tool designed to enhance the efficiency and effectiveness of TrakCare support advisors. This document provides a comprehensive overview of the project's development, from the initial proof of concept (POC) to ongoing refinements and future plans. Key insights and lessons learned during the project are shared to assist others in understanding the potential and challenges of implementing generative AI in a customer support context.

The technical feasibility of the project was straightforward to establish. The POC and a demo were rapidly created by a part-time team of volunteers who were balancing other responsibilities. The compelling results of the POC secured executive sponsorship for further development.

Developing the POC involved conducting user interviews to understand where the tool could fit into the workflow, to set parameters around suitable questions, and to focus initial efforts. These interviews also provided contacts for testing. Data protection and legal issues were carefully assessed, given the cloud-based nature of the intended AI tools.

The development process gave us experience with various tools and technologies, generating new ideas for additional approaches and use cases. We continue to develop Yoky and are carrying many of these ideas forward in other areas.

Who Should Read This Document

- Technical Leaders and Developers: Learn about the architecture, tools, and technologies used to build the Yoky Support Assistant, and understand the challenges and solutions related to integrating AI with existing systems.
- Customer Support Managers and Teams: Discover how generative AI can enhance support workflows, improve response times, and provide more accurate solutions to customer queries.
- Data Security and Compliance Officers: Gain insights into the risk review process, data deidentification methods, and legal considerations when using cloud-based AI services.
- Executives and Decision-Makers: Understand the strategic value of implementing AI in customer support, including the potential for innovation, user satisfaction, and operational efficiency improvements.

By reading this document, stakeholders will obtain a detailed understanding of the project's development phases, the lessons learned, and the considerations necessary for successful AI integration in customer support environments. This knowledge will be valuable for planning, executing, and refining similar AI initiatives in their organizations.

Introduction: The Idea

The project began simply as a creative idea. Our TrakCare support advisors and implementation teams were struggling with the vast amount of content (from documentation to service tickets and other internal sources). Our internal search tool had breadth of coverage but did not connect the dots from area to area. What if we could harness generative AI to analyze and search this content, providing better answers earlier?

This idea was submitted, along with many others, to an innovation program within InterSystems. A small team of part-time volunteers chose a name (*Yoky Support Assistant*) and developed a concrete pitch, which our executives then approved. The volunteer team then developed the proof of concept (POC) and a demo, within a short timeline. After the POC, the team (now with slightly different personnel) further developed the code, which is now in experimental use, and additional work is planned. This paper describes the process we've been through, and it shares our learning experiences.

Phase 1: The Proof of Concept

The first goal was to develop the POC of the Yoky Support Assistant. Early in this phase, we performed user research, tools research (not specifically discussed here), and our standard risk review. Then we worked on deidentifying the data we wanted to use. With that data available, we built the POC, which we then tested. The results were good enough to continue with a second phase.

User Research

We interviewed users from the groups we had in mind (all within the TrakCare organization), primarily support advisors and implementation staff members. The goal was to learn several things:

- What the users expected from a generative AI tool
- The user workflow and typical tasks (which would give us a feel of where a generative AI piece could be added)
- The kinds of questions that the users would ask a generative AI tool
- The data sources that the users typically consulted and most valued

We found that our prospective users had a diverse set of needs and expectations, not all of which could be addressed in a proof of concept, and consequently it was necessary to focus on a target subset. Considering the available data sources, the user workflows, and the user expectations, we identified the best target group as the TrakCare support advisors. These users work with an internally developed support ticketing system called iService.

After identifying the target user persona, we made the following key decisions:

- **Data sources:** A feasible starting place would be iService support tickets and the TrakCare documentation. Specifically, we ended up using a subset of deidentified iService tickets and a subset of English-language TrakCare documentation. (During risk review, discussed ahead, it became clear that

we had to remove personally identifiable information [PII] from the iService tickets.)

- **Types of questions we *could* answer:** Generative AI is a broad area. To evaluate the success of this phase, we identified types of questions that advisors often encounter and that are a good fit for the use of generative AI. These questions have forms like the following examples:
    - *What can you tell me about this subject from the documentation?*
    - *Have we encountered this error before?*
    - *Is there an existing root cause analysis for this error?*
- **Types of questions we *could not* answer now:** We identified other types of questions that cannot be evaluated yet because they either require data sources that are currently out of scope or that require PII. This boundary enabled us to properly set expectations with the users and keep the testing focused. Here are examples of such questions:
    - *Who is the InterSystems Account Manager for TrakCare General Hospital?*
    - *Are there any RCA (Root Cause Analysis) tickets for this specific customer?*
    - *Is there any Jira related to this error/issue?*
    - *Who on the UK Product Support team would be best to resolve this kind of issue?*
    - *What changes have been made recently in the product (Jiras, release notes) in a specific area/domain?*
    - *How does this functionality work in other regions?*
    - *Has this organization encountered this type of issue before?*

Risk Review

An initial part of the project included a standard risk review, focusing on information security, particularly because we had planned to use cloud-based services (Google Cloud Platform and many of its services).

The risk review was an in-depth process with our internal information security team and with Google to address concerns and questions about the handling of InterSystems data in the process of training the LLM. The chief concern was the potentially sensitive data, because the iService ticket data could inadvertently disclose the identity of business partners and proprietary business processes. This review included examining:

- Intended overall architecture of the POC
- Data required for the POC
- Use case for AI/Generative AI
- Cloud service components needed
- Security of the connections among the components
- Process to construct the POC

Our internal Legal and Security experts considered the project's compliance with corporate standards, and the approval process involved discussions about data security and the use of proprietary LLM technology. We also met with Google representatives and received the assurance that Google won't use InterSystems' data to train Google's AI models and won't store data longer than necessary to provide generated output.

Deidentifying the Data

We began the project with a large dump of data, including tens of thousands of iService tickets, as well as the English-language product documentation. The product documentation does not contain anything sensitive, but the iService tickets do sometime include personally identifiable information (PII).

Before feeding any of this data to a cloud-based service, we had to redact the PII. The first step was to evaluate an open-source Python module (the Stanford-Penn deidentification algorithm). We ran the module against the support tickets and then manually double-checked 1000 randomly selected support tickets. The evaluation was satisfactory for English-language support tickets.

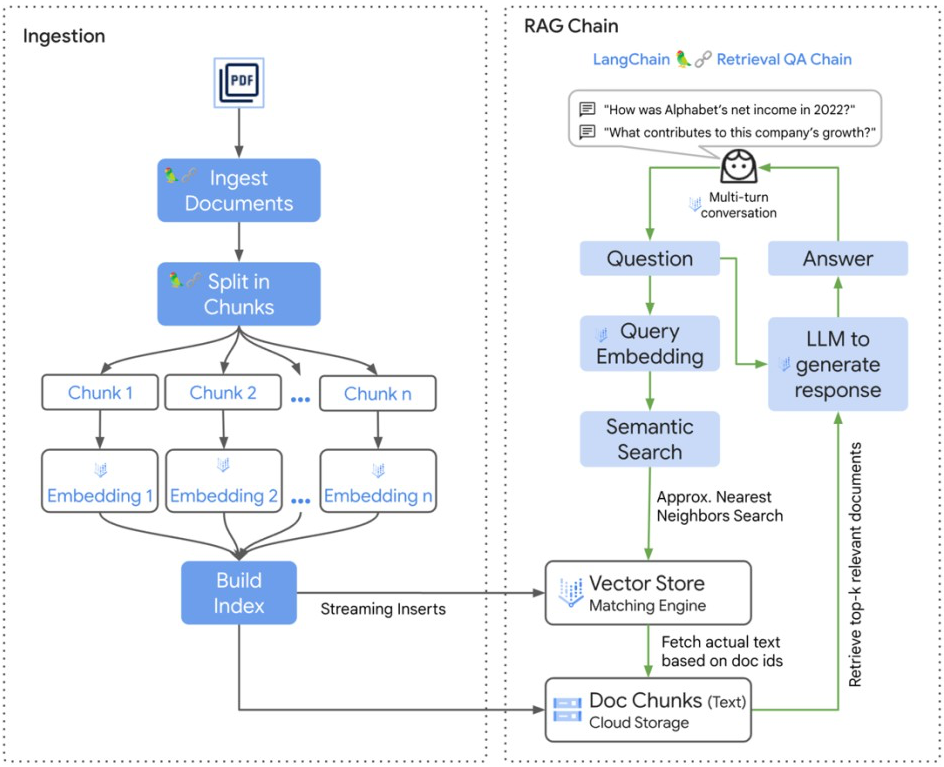
Building the POC

For phase 1, with our initial efforts focused on feasibility, we took the approach of starting simple with an out-of-the-box tool from Google, to see if this would efficiently achieve our objectives. While we achieved important advances, we did need to switch to a programming approach. Here is an overview of that evolution:

1. First, we tried Google's out-of-the-box tool Gen App Builder for ingesting and searching enterprise data using GenAI (generative AI), or what they term "semantic search with follow-up." The tool accepts data in various formats and then provides a simple search bar query and uses GenAI to do semantic search. While this tool does not provide the full chatbot experience, it seemed a reasonable place to start. We used it to develop a basic search application that used the product documentation as input.
2. With a primitive prototype working, we modified the iService ticket system so that some of our internal users could use this prototype from within that system, for test purposes. This was the first version of the Yoky Support Assistant. In the process of making this addition, we implemented secure OAuth2-based connectivity between InterSystems resources (behind the firewall) and Google's network. This connection enables us to send an encrypted query to the Google API and receive the results for display within the support ticket system. We also integrated Yoky with the InterSystems identity management infrastructure, thus ensuring that only authorized users could work with it.
3. Despite the useful advances we gained with Gen App Builder, we had to pursue an alternative path. At the time when we used this tool, there were limitations on the types and format of input data, and it was not feasible to index and search the documentation and the support ticket data together. As a replacement, we tried another beta-version tool, Vertex Generative AI Studio, which provides a nice user interface for creating and testing prompting techniques. This tool is closer to our vision for Yoky, and it has the option of connecting its output to a custom data source. Unfortunately, at the time when we tried this tool, there were gaps in its capabilities, and we were unable to use it successfully for our use case.

4. Our technical contact at Google suggested that we switch to a fully programmatic mode of development because that was the most flexible and mature part of their GenAI offering and because we would eventually need some programming to achieve our goal anyway. The same contact directed us to a sample that we could use as a starting place. We examined the sample, found it too complex, and then found another sample (also published by Google). The sample we chose uses Jupyter notebooks for interaction, which gave us an uncomplicated way to test our work. (On the other hand, at that time, it was not possible to call the relevant API from iService, so it would be necessary later to revisit that connection.)

At the end of this phase, the architecture included the standard data ingestion and Retrieval Augmented Generation (RAG) chain components, a solid foundation for future efforts:



Data ingestion includes different handling for our two sources of data. First we obtain the TrakCare product documentation in PDF format and upload it to Google. Google then parses it, runs it through the LLM, and indexes it. For the iService ticket data, we generate the deidentified data in the form of a CSV file, convert that to a Parque file, and then load it to BigQuery.

On the RAG side, we used LangChain, an open-source orchestration framework, to build a simple chatbot. To discourage "hallucination," we provided a prompt (see appendix A) that instructs the LLM to use references for its answers and to not attempt an answer that cannot be determined by the context alone.

For reference, phase 1 of this POC used the following technology:

- Google Cloud Platform

    - BigQuery
    - Vector Store and Matching Engine
    - text-bison LLM (in phase 2, we tried other LLMs)
- LangChain
    - GCSDirectoryLoader
    - BQTableLoader
    - RetrievalQAChain

Testing

The scope of testing in phase 1 was to assess the tool's capabilities in enhancing search functionality. In each round of testing, we worked with 2 or 3 participants, including support advisors, implementation specialists, and a sample of external customers who represent diverse user profiles. We provided a test harness (based on Jupyter notebooks) for participants to enter queries and assess responses from the tool, with opportunities to refine queries to look at ways to improve responses.

The aim of this testing was to:

- Test the accuracy, relevance, and comprehensiveness of the search results.
- Assess the tool's effectiveness in providing timely and helpful feedback and suggestions.

The test methodology was to:

- Conduct varied search queries to measure the accuracy and depth of the search results.
- Simulate scenarios for feedback during ticket submissions and proactive suggestions, evaluating relevance and usefulness.
- Track response times, accuracy and relevance of search results, and appropriateness of suggestions.

Test success criteria was based on user experience and comparing results to other tools, in particular our internal search system. The results were reasonable, but not uniformly so.

As an example, one query was "Can a TCI be cancelled and wait list be put back to initial?" Yoky found multiple places in the TrakCare documentation on this topic and then summarized as follows: "Yes, a TCI can be cancelled and the wait list can be put back to initial. To do this, the user must first cancel the TCI using the Cancel Pre Adm menu. Once the TCI has been cancelled, the user can then remove the waiting list entry using the Remove/Reinstate screen."

PHASE 2: REFINEMENTS

With phase successfully completed, we invested more seriously in phase 2, pulling in more technical staff and focusing on these areas:

- Improving the user experience, including tuning the results
- Revisiting the architecture

Improving the User Experience

After the POC, we began moving towards a more robust, chatbot interaction. This entailed several kinds of changes:

- Trying additional LLMs. We had tried text-bison LLM in phase 1. In phase 2, we tried Gemini, which was an improvement. By the end of phase 2, we were using Anthropic Claude Sonnet.
- Using an agent within LangChain so that a user can interact within a session, where conversation history is kept and used in later responses within the same session. As a result, the user can drill down into the details and follow up in various ways. (See appendix B for the prompt.)
- Processing the iService tickets before feeding them into the system. We had noticed that any LLM had a hard time understanding iService tickets that had long histories with many actions. Also, many tickets contain greetings, sometimes lengthy, between the customer and the support advisor; that information, even though deidentified, is not relevant to the kinds of questions we wanted to answer. We addressed both problems by generating a summary for each ticket and pushing it to Google along with the actual ticket data. We use the LLM to generate the summaries and also translate them to English if necessary. (This does mean that we are using an LLM during the data ingestion phase, in addition to later during the RAG chain phase.)

Revisiting the Architecture

Now that we understood more fully the processes of feeding the data to an LLM and processing queries, we realized that an interoperability production was a natural way to automate all this logic. We created a new production, with BASE, LIVE, and TEST environments, for this work. This production now handles both the data ingestion logic and the RAG chain logic, and all external connections are secured in the same way as in phase 1.

First, for data ingestion, this production extracts data from the source systems, transforms it as needed, and pushes it to Google for use by the LLM. It was seamless to include a call out to the same Python deidentification module we had previously used. It was simple to create business rules to handle the data. We can easily see the data at all stages of processing, and we can detect and correct problems. For the RAG chain logic, we were able to call all the necessary components from within the production, thus enabling us to see the state of processing at any point, with the ability to see error messages in sufficient detail for debugging. We knew that, for the user experience, we needed the ability to treat our sources of data independently, and when we moved the RAG chain logic into the production, we took that opportunity to make that code more modular.

Also, within the POC, we had added the Yoky Support Assistant page to the iService ticketing system, but we also had temporarily disconnected that page from the LLM, because the connection was more challenging to set up with the programmatic approach we finally settled on. In phase 2, we reimplemented that connection and adjusted all the logic to fit within our existing iService architecture. All in all, this means that input from the Yoky Support Assistant page causes a REST request to the iService REST service, which

in turn ultimately sends a request to the production mentioned above. The responses follow the same path in reverse.

Future Phases: Ours and Yours

Yoky Support Assistant is still in an experimental state but is undergoing further changes. The current ideas for improvement include these:

- Adding more data sources. One of the most cumbersome aspects of this project has been the requirement to scrub PHI/PII from our data sources before uploading to less-controlled cloud services. If we can keep data behind our firewall, we would be able to ingest more sensitive data sources including JIRA tickets and source code. Therefore, our future architecture will be hybrid cloud + on-prem – we will move the vector search to InterSystems IRIS so the data storage and indexing will be local, while continuing to leverage cloud-based LLMs for response generation.
- Improving performance. The competition in the LLM space revolves around performance including speed, and the latest GPT-4o and Gemini Flash models are several times faster than previous version, and Yoky will naturally take advantage of those advancements. Additionally, RAG systems can be improved by leveraging open-source models that are smaller, faster, and can be run locally. We plan to explore this avenue, especially as we consolidate more of the architecture to IRIS as described above.
- Scanning newly created iService tickets and using Yoky to pre-populate them with summary information and links to relevant documentation or related tickets.
- Expanding Yoky to also screen for possible clinical safety issues, which are not always immediately apparent to the support advisors.
- Considering other types of questions, based on newer data sources.

Yoky will remain an internal-use tool for the foreseeable future, invisibly helping our TrakCare customers by helping our support advisors with their questions and problems.

That said, we hope that sharing these project ideas and details inspires you to consider your own opportunities for using Gen AI. For help getting started, please see two of our newest learning paths (including assessments and badges):

- Getting Started with Generative AI (2h 45m) Learn about generative AI and how it differs from other types of artificial intelligence. Get an introduction to best practices for interacting with generative AI, and see how to use it safely and responsibly. Then, see how your own data can be used to augment the results of generative AI.
- Developing Generative AI Applications (2h) Build on the basics of generative AI by mastering retrieval-augmented generation (RAG), which enables you to create intelligent applications with improved accuracy. In this path, you will learn about concepts such as vectors and embeddings, and leverage Python libraries for practical RAG model implementations.

ACKNOWLEDGEMENTS

The team consisted of Baris Bekdas, Thomas Dyar, Maria-Sophia Fedyk, Dominic Green, Felicity Hodder, Melia Hughes, Alki Iliopoulou, Robert Luman, Otto Medin, Enrique Portela, Luke Ridding, Caroline Saavedra, Reneé Sardelli, Chris Stewart, and Helen Taylor.

The team thanks the InterSystems Innovation Acceleration program for sponsorship. The Yoky project is a direct outcome of this program.

The team also thanks the executive sponsors: Jeff Fried and John Paladino.

APPENDIX A: PROMPT USED IN PHASE 1

```
SYSTEM: You are an intelligent assistant helping the users with
their questions on InterSystems TrakCare Electronic Patient
Record system.


Question: {question}


Strictly Use ONLY the following pieces of context to answer the
question at the end. Think step-by-step and then answer. Within
the answer, summarize the references and provide some explanation
of how the reference helps to answer the question. If the
reference document is a pdf, it is from TrakCare Product
Documentation and should be used as a priority for answering
"how-to" or "can I" types of questions. Otherwise, the reference
is from a previous support ticket.


Do not try to make up an answer:

If the answer to the question cannot be determined from the
context alone, say "I cannot determine the answer to that."

If the context is empty, just say "I do not know the answer to
that."


=============

{context}

=============
```

APPENDIX B: PROMPT USED IN PHASE 2

Note that we used different LLMs in phases 1 and 2, so a direct comparison of these prompts may not be appropriate.

```
You are Yoky, a chatbot assistant for support personnel at
InterSystems, helping customers of the TrakCare EMR product. Your
primary role is to assist support personnel by providing concise,
accurate information and references from the knowledge base,
which includes documentation and a support ticket database.

Here is a record of your interaction so far:
\n\n{chat_history}\n\n

You have access to the following tools:\n\n{tools}\n\n

Formulate a response that directly addresses the query with
relevant details and insights. Your final answer should be a
clear, actionable recommendation or conclusion. If the input is
unrelated to InterSystems technology or you cannot provide any
relevant answer, respond conversationally as a human assistant
would.

Follow this exact format, always concluding with a Final Answer:

Question: the input question you must answer

Thought: consider what action to take

Action: choose an action from [{tool_names}]
Action Input: specify the input for the action

Observation: report the result of the action
... (repeat Thought/Action/Action Input/Observation up to 3 times)

Thought: I now know the final answer
Final Answer: provide the final answer to the original input question

Begin!

Question: {input} Thought: {agent_scratchpad}
```