SMU Data Science Review

Volume 8
Number 1 *Spring 2024*

Article 5

Enhancing Customer Support Operations through GPT & Q-Learning: A Model Study

Adam Alidra
Southern Methodist University, adam.alidra3@gmail.com

Bob O'Brien
Bob.OBrien@microsoft.com

Dalton Young
dyoung@microsoft.com

Enhancing Customer Support Operations through GPT & Q-Learning: A Model Study

Adam Alidra[1], Bob O'Brien[2], Dalton Young, PhD.[2]
[1] Master of Science in Data Science, Southern Methodist University,
Dallas, TX 75275 USA
[2] Microsoft, 1 Microsoft Way,
Redmond, WA 98052-6399
aalidra@smu.edu | adalidra@microsoft.com

Abstract. "Growth strategies that are purpose-led, customer-centric, experience-driven, data/AI-enabled, and technology-scaled require new mindsets…" (Cornfield, 2021). What can we take from this? Business growth and customer experience are inextricably tied together. Therefore, thriving, as an organization, is dependent on reimagining enterprise operations through modern, scalable data and AI technologies. Our study aims to enhance support operations with emerging AI capabilities, including OpenAI's LLM models, built on self-attention mechanism transformer architecture, and tailored for business needs through prompt engineering. Our research uses Markov Decision Process and the Q-learning algorithm to evaluate synthetically created support incidents. Through this set of methods, our study seeks to determine the optimal policy to apply for each incident, including demarcating low-cost self-service approaches, in which an agent leverages AI tools to support a support ticket resolution process versus following a traditional, resource-driven approach wherein higher-level expertise intervention and escalation is required. Our research also explores different aspects of AI model development and performance, including grounding data for content relevance, breadth of user intent, and the quality of user prompts, aspects which are fundamentally enabled through prompt engineering methods. Ultimately, in this analysis, we aim to elevate the support experience for Microsoft customers, reducing support staff burnout, while providing a blueprint for other businesses to improve support operation costs, and thereby, their bottom lines.

1 Introduction

Corporate strategies are driven by three primary motivations: revenue growth, driving operational efficiencies, and risk mitigation. The connective tissue between these drivers is the 'customer'. Changes in buying behavior and vendor expectations have pressured organizations to consider new and more effective ways to expand their market share while preventing customer attrition,

two key performance indicators used to gauge overall business health. Modern customer expectations are so palpable, in fact, a recent study found, “67% say their expectations for customer experience are at their highest,” while, “over 50% of customers will switch to a competitor after a single unsatisfactory customer experience,” (Cohen, 2023). The contemporary buyer-vendor dynamic has necessitated businesses to develop a multi-modal, omnichannel means for product and service-related inquiries to be addressed.

Historically, organizations and their customers relied on live agent calls to, respectively, deliver or receive product and service support, a manual, highly personalized yet expensive way to remediate support incidents. With the advent of internet and mobile technology, followed customer websites, online knowledge bases, and FAQ web pages. More recently, with artificial intelligence’s (AI) meteoric rise to near ubiquitous interest, enterprises are investing and adopting AI-driven chatbot solutions, more and more, to automate processes, scale their operations and meet customer needs and expectations including frictionless resolution.

These technological advances have broad implications for organizations. On one hand, a recent survey found, “Over 40% of users attempt to resolve their questions first by accessing online self-service support,” many seeing, “it as the easiest and quickest way to get an answer,” (Gupta, 2022). This supports the continued efforts we have already seen by organizations in meeting customer needs. However, building and maintaining a digital, automated support channel where service personnel and customers, alike, can readily search for documentation and resources grounded in relevance and accuracy has spawned new challenges underscored by some of the same, previous operational issues. Forbes cited, “companies see turnover rates of up to 45% of agents every year,” (Morgan, 2023).

While artificial intelligence, as a concept, dates to the early 1950’s, its earliest, rudimentary form, symbol-based algorithms, has since evolved to more scalable, sophisticated branches, including machine learning in the 1990’s, deep learning, and neural networks in the 2000’s, and most recently generative AI. Organizations are clamoring at the prospects of generative AI for various applications, including enhancing customer support experiences. A BCG article states, “As customer-service applications based on generative AI become more mature, companies will gain confidence in their performance, reducing the need for human oversight and allowing customers to interact with them directly,” (Bamberger et. al, 2023).

Analyzing regularly observed technology support incidents through machine learning frameworks and OpenAI’s GPT-4 model, we will form an understanding around what AI methodologies are most useful for enterprises to enable their support personnel to serve customers. In so doing, the expected

outcome is to enhance self-service support channels for enterprise customers. Specifically, we assess the support process comprehensively, by applying a reinforcement learning framework, known as Markov Decision Process, and applying Q-learning to our problem. Through this methodology, we also explore and document approaches to optimizing prebuilt foundation LLMs developed by OpenAI and Microsoft, including user intent, the underlying data used to ground the model, and the breadth of user prompts. This exercise has the potential to positively impact user experiences, helping customers realize better time-to-value of the products and services they invest in, reduce support agent burnout, while bolstering organizations' competitive advantage through optimized bottom lines and improved top line revenues.

2 Literature Review

The central objective of this section in our essay is to provide supporting evidence of how artificial intelligence, and specifically, generative artificial intelligence, has the potential to positively impact customers and the organizations they do business with, through an enhanced support experience. The literature review assesses whether adoption of generative AI technology has any bearing today on customer support operations and what methods, techniques, and user behaviors are likely to produce favorable outcomes for businesses. The review also introduces Markov Decision Process and Q-learning and its application in the real world, a central part of our research, results, and findings. While the motivation for this study is evaluating LLMs' feasibility in improving businesses' support operations, the review contemplates prompt engineering techniques including user inputs, known as prompts, including their intent and completeness, and the corpus of documents used to ground generative AI LLM models.

2.1 AI, Automation, and Customer Support

AI and automation, while unique subjects, have often been mentioned in causal terms. This notion, which has generally held true, has started to shift as industry pundits observe business leaders view automation from an individual empowerment lens rather than a resource displacement one. Where corporations previously aimed to identify processes to automate in the past with machines and technology, effectively reducing workforces or altogether replacing human workers, as AI continues to be widely adopted its macro use cases suggest businesses, "use artificial intelligence (AI) to augment human decision-making, problem-solving, strategizing and creativity… today's intelligent automation supports a business' overall market strategy," (Ghosh et. Al, 2021). Thus, instead

of humans ceding tasks to machines, AI is growingly being used to complement a worker's efforts, effectively extending human talents.

The appetite for adoption of AI and automation is motivated by various tangible opportunities for creating business value. These opportunities include mitigating business risk, improved customer satisfaction, higher revenue growth, and maximizing organizational efficiency. One case study illustrating the positive impact AI adoption had for companies highlights, "companies report that by implementing AI-driven knowledge base tools, they have improved first-contact resolution by five to seven percentage points, reduced handling time by 20% to 30%, and reduced new-hire training by 25% to 40%," (Ramachandran et. Al, 2020). This example demonstrates how several key desired business outcomes are achieved by leveraging AI. These include optimizing operational processes including cost per incident resolution for each customer inquiry, enabling workers to scale to effectively meet growing customer expectations, while arming them with the training and resources needed to be more productive. In inspecting what about leveraging AI in support channel is producing a return on investments for businesses, in Theresa Eriksson et. al's, "Think with me, or think for me? On the future role of artificial intelligence in marketing strategy formulation" purports, "AI not only enhances creative thinking but also supports context awareness, reasoning ability, communication ability and self-organization ability," (Malik et. al, 2021). This postulation, combined with examples, like the one above, supports the notion that AI use is increasingly having a symbiotic relationship with humans in the workplace, possibly even enhancing an individual's wellbeing.

Employee retention, another important metric for businesses, is typically correlated with employee satisfaction, or for purposes of this discussion, we will liken to wellbeing. A study on employee satisfaction found, "AI provides more flexibility and work-related autonomy by functioning as a complementary facilitator (21%)," (Malik et. al, 2021), based on moderately positive and very positive sentiment responses to the survey. While, at the outset, this sounds promising, Malik et. al follow with, "Organizations have benefitted in multifaceted ways… there has however been a flip side to these benefits." The authors enumerate unintended consequences, including, stress, social isolation, and the causal effects between workplace stress and health issues negatively impacting quality of life. They theorize these findings were not a result of leveraging AI, in and of itself, but rather the downstream effects of AI adoption, for example, "digitization of office work… increased workload… an overwhelming feeling of urgency, heightened expectations…" (Malik et. al, 2021). While these considerations should not be completely dismissed by organizations, the study points out the findings are confounded by general

insecurities that come with process changes in the enterprise, emphasizing the need for a phase-wise adoption and the appropriate upskilling so employees feel empowered rather than threatened- this is consistent with any other technology strategy, where people, process, and tools equally need to be contemplated in a rollout, to realize its optimal benefits. In summary, because support agents stand to have more freedom because of using AI to enhance their job function, they would perceivably have better job satisfaction. If this holds true, organizations are likely to see less attrition from their support personnel. These examples are illustrative of the virtuous cycle of AI including the benefits it unlocks for organizations.

2.2 Operationalizing Generative AI in Support Operations

AI's inception dates back almost 60 years ago, however, it was less than 10 years ago that the earliest algorithms which defined the beginnings of AI morphed into a broadly available set of models that are now deployed and scale, perform, and solve the demands of modern-day enterprise use cases. The introduction of neural networks, including generative adversarial network (GANs), represented the nexus to other neural network forms including recurrent neural network (RNNs), long short-term memory (LSTM), and graph neural networks (GNNs), to name a few. Most recently, readers recognize the prominent 'GPT' moniker, formally known as (self-attention mechanism) transformer models. Each of the mentioned neural network models are capable of handling sequence modeling and transduction problems, like language modeling and machine translation- in everyday terms, for example, these appear in the form of speech recognition, sentiment analysis, and recommendation systems.

The excitement for Transformers, and specifically self-attention mechanism transformer models, stems from the challenges they have overcome in respect to their predecessors. Each of the earlier neural networks aimed to address a limitation of earlier designed neural networks- for instance, RNN's, by design, process data sequentially, aligning input and output sequences to steps in computation time. In a RNN scenario, neural network, A, processes input x_t and outputs h_t, and loops the information so it can be passed to the next RNN step. Effectively, "RNNs can learn to use past information and figure out what is the next word for this sentence," (Giacaglia, 2019). While they serve their purpose in simpler scenarios, where the distance between relevant information and the place where that information is needed is in proximity to one another as the need for context increases due to the ambiguity and/or complexity of the user prompt, issues can arise, including inductive biases in the form of temporal and translational variances. Because information is passed at each step of the chain,

and as the input-output chain and relative information associated to the chain spans further, the probability of information being lost along the chain increases manifesting through temporal and/or translational variances, depending on the task at hand and expected user response. LSTM, an iteration on RNNs, makes small modifications, through multiplication and addition, to manipulate data into cell states- however, like RNNs, the probability of an accurate response exponentially decreases over space, producing similar results.

GPT, an acronym for Generative Pre-trained Transformer, and Large Language Models (LLMs), are neural network machine learning models trained to process sequential data, including, but not limited to, natural language text, time series data, and genome sequences. While transformers share the same autoregressive virtue of predicting subsequent words or phrases based on preceding words in a sequence, the dichotomy rests in how information is processed amongst these models and how the results are potentially affected. RNNs and LSTM compute inputs in a sequential pattern, whereas transformers can take an input sentence, commonly known as a prompt, and process it into a sequence of vectors, which are then encoded, then decoded into another sequence. What sets GPT transformer models further apart from previous neural networks is the self-attention mechanism, a critical piece in handling sequence modeling and transduction. The architecture follows the competitive neural sequence transduction encoder-decoder structures, although, "…the encoder maps an input sequence of symbol representations $(x_1,\ldots,x_n)$ to a sequence of continuous representations $z = (z_1,\ldots,z_n)$. Given z, the decoder then generates an output sequence $(y_1,\ldots,y_m)$ of symbols one element at a time, (Vaswani et. al, 2023). The key call out in this process is this flow is realized through the self-attention mechanism, which encapsulates dependencies across the chain of encoding-decoding pairs, to produce more accurate completions back to a user. To further contrast Transformers from previous neural networks, let us consider an architecture consisting of six encoders and six decoders. Each encoder, which resembles the next, is comprised of a self-attention layer and a feed Forward Neural Network. An input flows through the self-attention layer, where the message is encoded by word but also evaluated in relation to the other words in the sentence. The decoder works similarly, however, with an added layer called encoder-decoder attention, which enables it to home in on related parts of the input sentence.

With a general understanding of Transformer architecture, the self-attention process itself, like other natural language processing (NLP) approaches, begins with a concept called embeddings- in essence, this is the process of converting text into numerical values a machine or computer can interpret. Word embeddings are, "used to map words or phrases from a vocabulary to a corresponding vector

of real numbers," (Collis, 2017). The embedding process occurs in the bottom encoder, which then flows through the two layers, self-attention and Feed Forward. It is in this design and process where the juxtaposition between RNNs and Transformers becomes evident, the latter, inherently allowing each word embedding to flow independently through the encoder in a parallelized fashion versus sequentially. A sentence may be embedded into vectors, each word of the sentence represented through unique vectors. Vectorization is significant for two primary reasons, dimensionality reduction and contextual similarity. By embedding words and representing them in the form of vectors, the number of features or surrounding characteristics needed to represent words decreases, thereby reducing the amount of computational and memory-intensive resources required to process information, crucial for organizations looking to leverage advanced AI capabilities while keeping cost containment in mind. Additionally, word embeddings and vectors are a highly dense representation of, in this context, natural language, effectively capturing semantic and syntactic relationships between words. Hence, using vector representations as opposed to original natural language form has been shown to maintain the context similarity of words, thereby, expected to produce better results in use cases including reading comprehension, abstractive summarization, and textual entailment.

Comparing vectorized word embeddings and determining the applicable weight between their relationships, in the context of transformers, occurs through the attention mechanism- in fact, the mechanism computes information to attribute 'attention' between word embeddings, hence its name. There are different ways in which self-attention computes similarities between word embeddings, one of which is 'dot product', also known as inner product space (IPS). This computational measure starts by creating vectors from each of the encoder's input embeddings. Assume the phrase, 'I kicked the ball'- 'I', 'kicked', 'ball' are encoded as their own vectors. In this instance, 'the' is ignored as it holds little semantic meaning on its own and is instead combined with the adjacent words in the phrase, still encapsulating the relationships between the rest of the words. Given the example, we have three words, and therefore three vectors, each with a Query Vector, a Key vector, and a Value vector, abstractions we are using for the purpose of internalizing 'attention' as a concept- these abstractions are generated by multiplying the embedding, with an original dimensionality of 512, by a randomly selected number, three matrices. This is executed through a neural network layer called the linear projection layer, which produces a linear combination of input features, reducing the dimensionality of the vectors to 64. The idea, to reiterate, is to make the attention mechanism's computational process efficient and constant. Taking the embedding vector, represented as X_1 and multiplying it by the Query vector represented as q_1 we now have a matrix, W^Q.

We follow this same process for the Key and Values Vectors, resulting in matrices, W^Q, and W^K, and W^V. Next, the Transformer process calculates scores based on each word of the prompt sentence respective to the first word- this is produced by multiplying the query vector with the key vector of the word in scope, like ($q_1 * k_1 = 112$) to signify the first score, followed by ($q_1 * k_2 = 96$). Given the scores, the mechanism divides the scores by square root, 8, provided the dimensionality of the key vectors used, -64. Subsequently, using a Softmax operation, which normalizes the scores to amount to 1, we have:

Input 1: Score: 112, Divide by 8 ($\sqrt{d_k}$)= 14, Softmax= 0.88
Input 2: Score: 96, 96/8= 12, Softmax= 0.12

The Softmax score reflects the importance of each word respective to the position in a sentence. The word with the highest softmax score will correlate to the word at that position. Finally, by multiplying each vector by the softmax score the attention mechanism can sum up the weighted value vectors, for each word, starting with the first word input. During the input step of this process, it is worth calling out the importance of how the mechanism retains data regarding the order of words in a sentence and their positional significance- this is achieved through positional encoding. After these computations occur, vectors are then passed to the feed-forward neural network.

An iteration on inner product space, also used in transformer models to assess similarity between word embeddings and their respective vectors, is cosine similarity- in fact, AI services, like OpenAI and Microsoft Azure OpenAI embedding models rely on cosine similarity to evaluate the similarity between a user query and the corpus of documents used to ground or train the LLM, more on the concept of 'grounding' and Azure OpenAI models later. "Cosine similarity is a metric used to determine the cosine of the angle between two non-zero vectors in a multidimensional space. It is a measure of orientation and not magnitude, ranging from -1 to 1," (Porter, 2023). Like IPS, word embeddings are the sum of the attention, including weights and biases, learned by the transformer. Visually, picture a vector as a line from the origin or input to the location of a word in corpus of data encompassed within the model. Two vectors with semantic similarity will both point in similar directions, with an angle closer to '1'. The closer a value is to '1', the more it signifies a high similarity score, meaning the LLM solution has produced relevant information back to a user based on provided prompt.

There are advantages and disadvantages between the two measures, which fundamentally stem from the difference in how vectors are compared. IPS, like cosine similarity, evaluates the angles between vectors, however, also calculates

the magnitude of vectors, making it a quantitative measure. IPS vector similarity is based on the sum of calculations provided the earlier described computational process- however, if one vector is significantly smaller or larger than the other, IPS will skew towards the larger vector. Conversely, cosine similarity only focuses on angles, making it invariant to the magnitude of vectors, and a qualitative measure of word similarity. The academic journal, "Cosine Normalization: Using Cosine Similarity Instead of Dot Product in Neural Networks," written by Luo Chunjie et. al, argues dot product, "…is unbounded, thus increases the risk of large variance. Large variance of neuron makes the model sensitive to the change of input distribution, thus results in poor generalization," (Chunjie, 2017). The dot product of (a) word embedding(s) being unbounded means it can take any value between negative infinity and positive infinity, leading to a concept called 'overfitting', especially in longer sequences. In this scenario, the model becomes overly complex and fails to generalize for new inputs, thereby producing a less optimal result for users. Cosine similarity addresses this issue through normalization evidenced through the boundaries set, -1 to 1. Although cosine similarity is more robust in the context of magnitude compared to IPS, to obtain optimal performance, more training, like enterprise documents, may be required inclusive of the LLM.

The power and sophistication of how transformer models' function is impressive, although, equally of substance and merit is its accessibility, from the developer and IT professional planning to deploy these capabilities as enterprise solutions, to the user looking to improve their work function and general quality of life. For the user, this means using GPT models, like the prominently known ChatGPT, to create or summarize information, generate code, or rapidly search through a corpus of documents. GPT-like models, which have been trained by assembling massive amounts of publicly available data into a large, supervised dataset, are generic enough to be adaptable and solve a multitude of tasks. For end users, the strength of LLM capabilities lies in the simplicity of interacting with the service, made available through an easy-to-user interface, like a chat app or browser-embedded bot. The workflow for end users is straightforward, 1) a user enters a prompt (input), 2) the LLM processes the prompt through the billions (or even trillions) of parameters used to train the model, and 3) the model returns a completion (or response) for the user to then act on.

OpenAI and Microsoft have developed enterprise grade LLM services available to enable the workflow described above, including GPT 3.5 Turbo and GPT 4 Turbo. Fundamentally, these are API services based on a completions endpoint which grant access to the model's text-in, text-out interface. An example of this is:

Prompt: """ count to 5 in a for loop """
Completion: for i in range(1, 6): print(i)

Azure OpenAI's text-based models leveraged in this study convert text into tokens. "Tokens can be words or just chunks of characters. For example, the word "hamburger" gets broken up into the tokens "ham", "bur" and "ger", while a short and common word like "pear" is a single token," (https://learn.microsoft.com/en-us/azure/ai-services/openai/overview). The length of the prompt, completion, and any instantiated parameters dictate how many tokens are processed in any given request. There are three primary parameters available in the Azure OpenAI playground experience, prompt engineers can explore to reach desired performance of the model being deployed. The first, temperature, limits completions to be more deterministic, in essence producing more consistent and repetitive responses. The higher the temperature, the more creative and randomized responses become. The next parameter is Top P, working similarly to the temperature setting, by allowing for less probable responses to be included. The last parameter is tokens- coalescing our understanding of how Azure OpenAI tokenizes words to process a request, setting the token parameter governs the combination of sub-words used by the user prompt and completion, effectuating the throughput cost of using the service.

While foundational large language models ostensibly are acceptable for the general population, given the benefits of using the tool have been found to outweigh the absence of the tool, enterprises have quickly sought to customize general LLMs to serve the needs of their organizations, a pattern widely known as retrieval augmented generation (RAG). In parallel with serving a business' specific information retrieval expectations, RAG mitigates a risk generic LLMs are susceptible to, a concept known as 'hallucinations,' in essence, producing incorrect or nonfactual responses. Heidi Steen at Microsoft defines RAG as an, "…architecture that augments the capabilities of a (LLM)… by adding an information retrieval system that provides grounding data…. For an enterprise solution… you can constrain generative AI to *your enterprise content* sourced from vectorized documents, images…" (Steen, 2023). This architecture, at a foundational level, entails using an LLM model in conjunction with an organization's use-case specific data, for example, text from a corpus of documents or other unstructured data stored in an object storage container, or relational data from a database, to return enterprise context-aware information to a user, given their prompt, through a simple-to-use interface, like browser or chatbot embedded into an enterprise application. With respect to grounding LLMs, "it is crucial for ensuring the quality, accuracy, and relevance of the generated output," (Berger, 2023). Because LLMs are trained on data that is stale and public,

corporate data that is sensitive and/or proprietary will be missing without grounding employed. In addition, whereas data science historically focused on big data, a study conducted by Andrew Ng, from the MIT Sloan School of Management, "…argues that focusing on the quality of data fueling AI systems will help unlock its full power," (Bukowski, 2023). While grounding data has the potential to improve LLM performance, a poor implementation can produce similarly negative performance, again, emphasizing the need for high-quality data, both at the outset of a model deployment, but iteratively as business environments evolve, new information is published, and models are developed, to name a few variables that have potential for impact.

Sophisticated RAG designs, developed using model performance-enhancing techniques, like prompt engineering and/or fine-tuning, although more complex to implement and operationalize, are purported to increase performance, accuracy, and scalability for businesses. Chia Jeng Yang, author of "A first intro to Complex RAG (Retrieval Augmented Generation," surmises, "…most of the work that goes into building a RAG system is… adding additional contextual guardrails that allow the LLM to make more deterministic information extraction." By adding parameters and prescribing boundaries around generative AI model's completions to users, the ability for the model to produce a more literal output respective to the input, has a higher probability of occurring. An example of a sophisticated design, gleaning on our understanding of embeddings, involves embedding document chunks, storing the embeddings in a vector database, and similarly, taking a user's prompt, embedding the question to identify the correlated document chunk embedding, which is retrieved and processed through the LLM to produce a generated answer.

Although RAG designs can vary, given the complexity of an enterprise's use case(s) and performance requirements, they typically share several common traits across implementations. First, organizations must consider data preparation, including identifying the relevant data sources for their use case, and extracting the data. For instance, for data wrangling, AI engineers can leverage pre-built object character recognition (OCR) models for PDFs, or web crawlers for information on the internet that is programmatically parsed, effectively making the additive data usable. Subsequently, a natural language process known as 'chunking' occurs- because enterprise-specific GenAI implementations generally involve larger volumes of data to ground the LLM, chunking serves the purpose of segmenting data into smaller chunks that can efficiently and accurately be searched against and surfaced for user consumption. The chunking strategy is a critical piece in the scheme of RAG, where developers must balance capturing the context of information and the performance and cost effectiveness of the solution,

as larger chunks are more computationally intensive due to the noise it introduces as information is processed back to a user. One recent advancement to the chunking process is adding metadata to the chunks in the document processing stage, like, date of a reference, to facilitate information returned based on recency when asked by a user for the most recent- this exercise can also apply to other scenarios, like ranking based on importance, and grouping according to similarity or dissimilarities of a keyword or phrase input by user. Different approaches exist and continue to be introduced around chunking, with the goal of optimizing this step, and are deployed based on needs, including whether information is found on a single or multiple documents.

A couple other design elements that are regularly contemplated in enterprise RAG patterns include semantic search and vector indexes and databases. Semantic search, "…involves indexing documents or fragments of documents based on their semantic representing embeddings. During retrieval time, a similarity search is performed from the semantic representation of the query to find the most relevant documents," (Berger, 2023). In essence, semantic search works similarly to our detailing of transformer models, in that words are converted to embeddings, multi-dimensional numerical representations of "meaning". Provided the natural language prompt, the output is a vector consisting of anywhere from hundreds to thousands of numbers that are evaluated for similarity within a three-dimensional space. In so doing, organizations benefit from a more efficient and accurate search-driven AI-system. Vector indexes and databases, too, are essential tools when solving natural language processing problems, especially in reference to embeddings. "These systems store documents and index them using vector representations, or embeddings, which allows for efficient similarity searches and document retrieval," (Berger, 2023). Vector databases become increasingly important when scaling a RAG implementation. To bring these various aspects of RAG together, in this context, it starts with preprocessing documents by chunking them so we organizations can index more relevant information and due to the inherent context window constraints of the language model service. Chunks are then converted to embeddings that are calculated and stored into a vector index. When a user enters a prompt or instruction, the system processes or calculates the input, which triggers the similarity search to retrieve data within the vector database, that is most semantically like the query. Finally, the documents are ranked and curated as a response, according to query relevance.

It is evident that large language model enhancements, directly and by extension, are surfacing and evolving daily. To focus on a widely examined and adopted set of practices, amongst organizations deploying GPT-model solutions in their enterprises, is a relatively novel concept garnering broad attention known as

'prompt engineering'. "Prompt engineering involves carefully crafting the instructions or input provided to a language model… It includes selecting the right wording, structure, and context to guide the model…" (Sanchez, 2024). Prompt engineering aims to embed directions into various aspects of an LLM process to improve the likelihood of a relevant response to users. Prompt engineering can be facilitated through the actions of a user, like giving clearer instructions to the LLM-based task chatbot and asking for justifications when completions vary or consist of many possible answers. Justifications, in RAG, appear in the form of a mechanism called, 'attribution.' Source attributions allow, "… the model to indicate the origin of the information in the generated response. This transparency can enhance user trust by providing visibility into the model's decision-making process," (Ani, 2023). Attribution is a critical aspect of operationalizing generative AI in the enterprise, as business users can review how a completion was derived and verify its origins are authoritative, in nature. To adapt a foundation model to serve a specific task, developers building an LLM-based solution, may employ the use of zero or few-shot learning to exploit the prompt-completion structure. In practice, prompts affect the model weight configuration with the goal of completing the desired task. The distinction between the two learning methods is zero-shot learning prompts preclude examples of the correct output, only articulating the task description, whereas one- or few-shot learning demonstrates an input and a hand-crafted example or exemplars, respectively, of a correct response. For example:

Task description: Translate English to French
Examples: sea otter => loutre de mer ; plush giraffe => girafe peluche
Prompt: cheese =>

The same study used to illustrate the above example analyzed the efficacy of zero-shot and few-shot learning, observing, "While zero-shot performance improves steadily with model size, few-shot performance increases more rapidly, demonstrating that larger models are more proficient at in-context learning," (Brown et. al, 2020). The term 'in-context learning', synonymous with few-shot learning in this study, describes the process in which a language model learns in-context learning sequences based on task-specific text input. The same study also suggests few-short learning consistently performs better than zero-shot and one-shot learning models- hence, exploiting the LLM with, "*K* examples of context and completion, and then one final example of context," (Brown et. al, 2020) has the highest probability of completion accuracy respective to the three learning scenarios. The performance of few-shot learning on GPT-3 was also evaluated against a language model enhanced through another technique, called fine-tuning. This approach, "…involves updating the weights of a pre-trained model by

training on a supervised dataset specific to the desired task… the main advantage of fine-tuning is strong performance…," (Brown et. al, 2020). While fine-tuned models, sometimes called state-of-the-art (SOTA) specialist models, have their qualities, and have been shown to outperform earlier language models, like GPT-3 in certain use cases, they have several disadvantages- Brown et. al contend, "… to achieve strong performance on a desired task typically requires fine-tuning on a dataset of thousands to hundreds of thousands of examples specific to that task." This statement gives rises to concerns around scalability and the cost-to-performance ratio. The practical considerations of fine-tuning mean the developed language models have specific, and thereby, limited applicability for different use cases.

To further support the argument generalist foundation models can achieve similar or better results when few-shot learning over state-of-the-art (SOTA) specialist models, a more recent study analyzed MedPrompt, a composition of several prompting strategies. Used in conjunction with GPT-4, the latest OpenAI model with a training dataset of 1.76 trillion parameters, or roughly, a 10x increase from its predecessor, GPT-3, the study postulates, "Steering GPT-4 with Medprompt achieves a 27% reduction in error rate on the MedQA dataset (USMLE exam)… we should the power of Medprompt to generalize to other domains…" (Nori et. al, 2023). Medprompt represents an amalgamation of prompting techniques including in-context learning (ICL), and two new concepts we will introduce in this study, Chain of Thought (CoT) and Ensembling. "CoT… employs intermediate reasoning steps prior to introducing the sample answer. By breaking down complex problems into a series of smaller steps, CoT is thought to help a foundation model to generate a more accurate answer," (Nori et. al, 2023). An example of this looks like:

Task: "Break this problem down step-by-step"
Input: "When I was 6 my sister was half my age. Now, I'm 70. How old is my sister?
Exemplar Step 1: 6 years old, my sister is 3 years old
Exemplar Step 2: 7 years old, my sister is 4 years old
Exemplar Step 3: 8 years old, my sister is 5 years old
…
Prompt: 70 years old…

ICL and CoT are symbiotic, often used together to form few-shot demonstrations. Moreover, GPT-4 was shown to be capable of, "autonomously generating high-quality, detailed CoT prompts," (Nori et. al, 2023). Hence, based on this argument, with the use of GPT-4, one can generate a series of prompts to

solve complex problems, which can then be reinputted into the few-shot learning process prior to a model deployment and be used to improve the accuracy and effectiveness of an LLM-based solution for support agents. This is an important distinction to the SOTA model, Med-PaLM 2, evaluated in the study- where CoT examples were hand-crafted by clinical experts with the SOTA model, GPT-4 with Medprompt used CoT rationales generated by GPT-4. The result from the latter approach, sometimes called program-aided CoT, includes rationales that, "…are longer and provide finer-grained step-by-step reasoning logic… recent works also find that foundation models write better prompts than experts do," (Nori et. al, 2023). Thus, by deriving CoT instructions with GPT-4 and using those steps in the few-shot learning process, the purported outcome is improved accuracy. Although, the combination of ICL and CoT have shown promise, CoT has been shown to be susceptible to 'naïve greedy decoding' when solving complex problems- in the example above, CoT constructed a response of '35', based on the locally optimal choice. Ensembling aims to address this hallucination risk through a technique referred to as *self-consistency*- by taking outputs of multiple model runs and using a, "…sampling method to produce multiple outputs that are then consolidated to identify a consensus output," (Nori et. al, 2023). In summary, by sampling multiple, diverse reasoning paths through few-shot CoT, ensembling takes the generated outputs and applies functions, like averaging, consensus, or majority vote to reach the most consistent answer. This technique improves upon standalone CoT, however, comes at the cost of increased computational demands, and therefore costs to a business.

2.3 Optimizing Business Operations through Data Science

Data science and traditional machine learning approaches, even with the advent of generative AI and its exponential adoption in the enterprise, still lends a crucial role in enterprise technology strategies, including complementing out-of-the-box foundation models. The same reasons a data scientist may look to control training and test data, set hyperparameters, identify bias, and monitor models in production, apply in the context of leveraging LLMs as well. One way data science has been employed in businesses, to analyze existing support processes and identify patterns for improvement, is through reinforcement learning, a branch of machine learning. Customer service and support operations are inherently complex domains, due to the dynamic nature of interactions between customers and agents. Support process actions are contingent on the situation and other variables, like the product/service, in question, including any service level agreements (SLAs), the level of support the customer is potentially entitled to, and/or the value a company places on that customer, for example, as a VIP. Hence, reinforcement learning (RL) models have been used to learn through trial-and-

error process guided by continual feedback – based on the observed state and the subsequent step taken, the agent receives a reward or penalty, provided the outcome. Then, the agent updates its approach recursively to optimize for expected future rewards.

This study utilizes Markov Decision Process (MDP), a mathematical framework for modeling sequential decision-making problems, and Q-Learning, a reinforcement learning algorithm, to evaluate a dynamic and randomized set of customer support scenarios. This strategy was chosen due to the absence of benchmark data available, relative to current support operations metrics at Microsoft, and because generated support tickets can often be nebulous, either due to the user's entry, the inherent complexities of a large enterprise technology landscape, and/or a multitude of other unrealized variables. MDP is comprised of states and actions, that define the environment, transition probabilities, which constitute the propensity of actions moving across states, rewards that quantify an outcome, and a policy, a procedure for selecting actions in states. The Markov assumption reads, "The future is conditionally independent of the past, given the present… we assume that successor states are conditionally independent of all states and actions that took place prior to the last state. This assumption is wrong. But useful." (Piech, 2013). The assumption suggests the successive state of the system only hinges on the existing state, and not on the sequential events that preceded it. In mathematical terms, this statement is expressed as:

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots\ldots, S_t]$$

The current state of the agent is denoted as S[t] and the next state, represented as s[t+1]- effectively, under this assumption, the assumption suggests the current state encapsulates information from preceding states. However, the excerpt states the assumption is false- this is because, in practical terms, the future state of a system could depend on its preceding states. Despite this known observation, MDP proves purposeful in general scenarios, including this study, as a starting point to understand and solve problems. Having said this, "there are generally two goals of inference…(1) Having an agent chose an action given a current state, (2) creating a policy of how agents should act in every possible state," (Piech, 2012). Applying this framework to our study, we take synthetically created data and execute an action based on the incident 'n', and create a programmatic process for the agent to follow with the goal of optimizing for lowest negative reward, or put differently, for lowest impact to business and/or customer.

For this study, a variation of Markov Decision Process coined 'partially observed Markov decision process (PoMDP)' is exercised. POMDP is, "… a

combination of a(n) regular Markov Decision Process to model system dynamics with a hidden Markov model that connects unobservable system states probabilistically to observations," (Hahsler et. al, 2021). POMDP diverges from classical MDP in that the agent's observations are dependent on the state even though it does not directly observe the system state, comprehensively. Hence, the observations are utilized to form a probability distribution, simply coined a belief state. This probability distribution is derived from a belief about what state the system is in, at that point in time, given all possible states. The POMDP framework is informed on what action to take based on (a) preestablished policy or policies, given the belief state(s).

Q-learning, a portmanteau for 'quality' and 'learning', is intrinsically a model-free, value-based, off-policy algorithm used to solve MDP- it works by identifying the most optimal sequence of actions based on the agent's current state. As a model-free algorithm, Q-learning is particularly valuable when transition and rewards functions are nebulous or altogether unknown- instead, Q-learning algorithm is an *a posteriori* method, meaning its learning occurs based on the outcomes of its actions. The value-based aspect, "…trains the value function to learn which state is more valuable and take action," (Awan, 2022). The value function, also known as the Q-value, is derived from each state-action pair and the expected cumulative reward probability, given a specific action is exercised in a particular state following an optimal policy. Q-Learning, generally, is an off-policy algorithm, which entails the model evaluating and updating policies that deviate from the original policy used to take an action, but nevertheless with the goal of discovering good policy given uncertainty.

This study parts from the generic Q-learning algorithm, specifically, the off-policy definition, instead enforcing an on-policy approach during the learning which takes place. Mathematically, this is represented as:

$$\pi(a \mid s)$$

The pi notation translates to the probability of taking action *a* given the agent is in state *s*, or more simply, the likelihood an agent will execute an action based on the real time scenario. To further draw the distinction from off-policy Q-learning, rather than updating a different policy, based on an action's outcome, the same policy is updated, and therefore, improved. This is realized, in our research, by employing an epsilon-greedy policy, allowing the agent to discover or explore the established state and exploit based on best action to take. "The concept of exploiting what the agent already knows versus exploring a random action is called exploration-exploitation trade-off," (Baeldung, 2023). In practice, the goal is for

the agent to discover its environment, take a trial-and-error approach provided available actions, and find the most optimal path to reward, or in this case, support resolution for each state in the environment. Exploitation can take place to realize a more immediate reward, even if it is not an ideal outcome- this illustrates the exploration-exploitation conundrum. It is worth highlighting, in a scenario where more data could be observed, in this case, robust details about a customer's support concerns, an established policy could be beneficial to seed a process called policy iteration, which is an alternative POMDP strategy- this approach deviates from Q-learning, however.

Through all this, the Q-Table, is the source of truth. Basically, it is a data structure of sets of actions and states which is updated as the algorithm 'learns'. States and actions are inputted into a Bellman equation, producing a Q-function. Mathematically, this looks like:

$$Q^{new}(S_t, A_t) \leftarrow (1 - \underbrace{\alpha}_{\text{learning rate}}) \cdot \underbrace{Q(S_t, A_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \Big(\underbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(S_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} \Big)$$

The equation illustrates how subsequent state-action values are intertwined into source state-action pair's value. Assume given state *s* action *a* is taken, where the state-action value pair is -50 reward- however, the model puts you in another state whose optimal action has a state-action value of -10. This indicates the -50 state-action value is closer to a -10 state-action. Therefore, the -50 value marginally increases in value given the model has explored and discovered a one-step link between the correlated state-actions. Conversely, if the learner takes an optimal action, subsequent to the previously mentioned -50 reward example, and this leads to a state-action worth -80 reward, the result is an additional negative reward which downgrades the -50 state-action to -60 or -70.

The equation shows how subsequent state-action values are mixed into source state-action pair's value. Say you are in a state s and take action a, that state-action value pair may be currently worth -50, but it puts you in another state whose optimal action is has an state-action value of -10. That means that the -50 state-action is close to a -10 state-action, so the -50 value increases in value a little as you've discovered a one-step link between these state-actions. But if you take an action and wind up in a state whose optimal action has a state-action worth -80 and you get a additional negative reward for doing that, your -50 state-action will be downgraded to -60 pr -70 or so. These abstracted examples are consistent with real

life, too, like in customer support, for example, where given an array of variable parameters, like the support agent's manager, may influence the reward potential across personnel.

Reinforcement learning, in general, has various real-life applications including for business operations. One such case study to illustrate its utility is in, "Real-time Departure Slotting in Mixed-Mode Operations using Deep Reinforcement Learning: A Case study of Zurich Airport," authored by Duc-Thinh Pham et. al. The premise of the 2021 study was to effectuate a mixed-mode runway operation to increase, "…the runway capacity by allowing simultaneous arrival and departure operations on the same runway." Like this study on support operations, the Zurich airport case study contended with a stochastic environment, in which arrivals, departures, and associated environmental parameters could lead to randomized scenarios, which need to equally be accounted for. The paper proposes, "a Deep Reinforcement Learning approach to suggest departure slots within an incoming stream of arrivals while considering operational constraints and uncertainties." This study, too, used Markov Decision Process and aimed to maximize the sum of future rewards. The reward mechanism was designed to achieve the highest utilization of the runway slots while considering safety and mitigating risks including violations of ICAO flight governing body. The results of the study supported the proposal reinforcement learning could be implemented to improve the mixed-mode runway operation, evidenced through the efficiency ratio of 83.8% for expected departure slots. This example is one of many observed across industries and use cases to corroborate the viability of Markov Decision Process and reinforcement learning as a valuable framework and algorithm, respectively, in instances where the state or environment of a problem is dynamic and variable in nature.

2.4 Literature Review Methodology

This literature review takes a methodical approach in laying the bedrock for subsequent steps of the study including our methods approach, results and findings, and broad, philosophical discussion points to consider for potential, subsequent complementary and supplemental studies. Bringing to bear our understanding of customer support and AI's proliferation in the enterprise space, we sought to discover research that, both, supported our belief AI has positive impact across business domains, including customer support operations, and identified any detriments caused to businesses and/or their end users due to AI adoption. To enforce objectivity, a range of broad industry perspectives were

reviewed and, in many instances, cited directly on the topics of enterprise customer expectations around support experiences and the impact of AI-enriched customer support operations. These references include widely respected consulting firms, like McKinsey, Deloitte, and Boston Consulting Group, to research and advisory firms, including Gartner and Forrester. The excerpts included in this part of the literature review were published between 2019 to 2023, accurately conveying the modern climate of customer expectations from support agents and the growing adoption of chatbot capabilities, and most recently LLM-models, in the enterprise.

The next section of the literature review encompasses various definitions, processes, and findings from reputable technical practitioners and authors, academic researchers, and technology providers, including Microsoft, OpenAI, Google, and Deep Learning- the criteria focused on understanding modern design patterns and technology, as it relates to the evolution that led to transformer model architecture, natural language processing vis-à-vis word embeddings and vectorization, and Microsoft and OpenAI's commercial GPT offerings and how they can fit into enterprise support. Also, newer concepts, like prompt engineering were introduced and explored. Given the first contemporary LLM was only developed in 2017, the material used to support this part of the literature coincides with the inception year of LLMs through 2024.

The literature review concludes with an examination of Markov Decision Process and Q-learning, as concepts and where they apply in real world scenarios. The resources brought to bear come from academic and research bodies, including but not limited to, Stanford and USA/Europe Air Traffic Management Research and Development. While there do not currently exist any case studies around customer support operations encompassing Markov Decision Process and Q-Learning, parallels can be drawn from the case study we represented towards a wide array of use cases, including improving support processes in the enterprise.

3 Methods

Depicted in Figure 1 is a high-level illustration of the methods applied in this research paper.

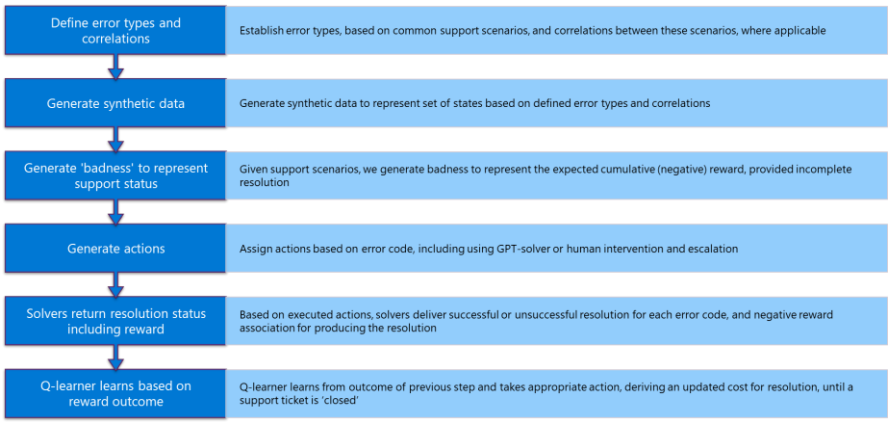

Figure 1. Q-Learning Method Process

3.1 Define Error Types and Correlations

Our analysis commences by encoding common support incident syndromes in binary form, effectively defining each error type and their respective correlations. It is worth highlighting that our process above is recursive as the algorithm 'learns' based on the reward outcome, consequently regenerating actions until an appropriate resolution is reached. Figure 2 enumerates the state types.

1. User authentication error
2. System latency
3. Library upgrade version incompatibility
4. Data format incompatibility
5. License expiration
6. Communications buffer overflow
7. An application segmentation fault
8. Out of memory

Figure 2. State Types

The state types indicate single-points-of-error, which may be involved in an error syndrome. Recall, when employing Q-learning, there are states, actions, and rewards. In this study, the customer support incidents equate to the states, a resolution occurs via an action or actions, which generate a reward or rewards, respectively. Each error state is represented by a combination of bits, where each bit corresponds to a possible failure or, in general terms, customer support incident. The error states are binary, meaning each component of the error type can fail or not fail, and error states signifying multiple error bits are the amalgamation of the individual failures, although a compounded set of issues may have even worse 'badness' than its individual parts. Conversely, some syndromes can be considered trivial, thereby not warranting immediate remediation. Error states can be decomposed into simpler failure modes, such as single bit errors or compound errors. Hence, incidents can, both, be independent and accrue to a separate compound error, where the underlying incidents may be causal in nature- the latter scenario is an example of a correlated incident.

3.2 Generate Synthetic Data

The aim in this step is to generate synthetic data that represent isolated symptoms and coalesce those same symptoms into more complex support issues. Pursuing our problem in binary terms, 5 error bits or 32 error states are produced, numbered from 0 to 31. 0 is the null error, or non-event, and 31 signifies the worst incident type, effectively, an aggregate of multiple issues and potential actions required to remediate the support ticket. To engender a more realistic action state, a Toeplitz matrix is employed, effectuating correlations between the bits so some errors have a higher probability of taking place together, in essence manifesting a conditional scenario, where one error is likely to happen as a symptom of another error. To best generate correlated binary random variables, in this instance, marginal probabilities of +1 were set to 50% for all bits.

Figure 3 depicts the error states, based on the executed correlation steps. It demonstrates that errors occur in an other-than-uniform frequency and without correlation one would instead observe, simply, a flat line. Also, given *a priori* knowledge does not exist to inform the model on what and how to learn, while the frequency of the bits is outside the scope of this study, in a realistic, empirically driven model, error bits would occur with different frequencies, which would be analyzed accordingly.

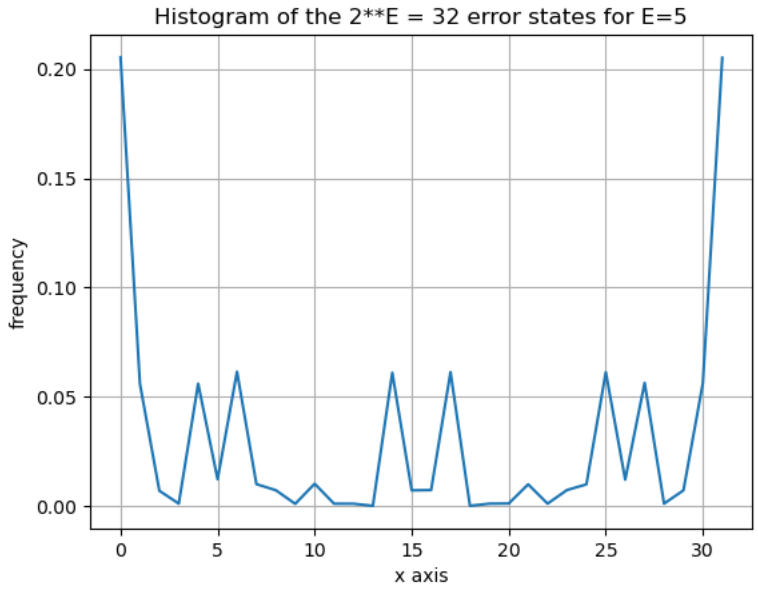

Figure 3. Distribution of Error States

A GPT model is employed to generate synthetic error reports based on error states. The synthetic error reports can then be entered as instructions to a GPT service, and a new prompt can be used to extract the apparent error modes from the reports. Using GPT to, both, encode the random correlated errors as quasi-open-ticket-text and subsequently decode the error bits from the same self-encoded text is a way to further obfuscate the message, thereby producing an even more "partially observed" error bit or ticket generation from users. Most importantly, it is enough that users report only the most egregious symptoms of their syndrome(s) for us to carry this study forward.

Although we could not extend our research to also model against previously referenced prompt engineering techniques, relative to user intent, breadth of response, and grounding of foundational models given a lack of Microsoft-specific support data, it is noteworthy to mention how modeling around these variables would apply in this step and our broader method process. Observe in Figure 4, a customer email is entered, which was also generated by an LLM based on instruction, as background information, along with a prompt that asks an Azure OpenAI model to identify support-related entities. Displayed on the left is a rudimentary instruction provided to the LLM, which includes a list of support types, like the list from Figure 2. With real life customer email examples, and iteratively refining instructions, which could be generated by an LLM, and

inclusions of few-shot learning examples, the aim would be to improve the foundation model. Furthermore, by incorporating knowledge base information into the LLM, effectively specializing the model to enable a self-service channel, for customers and support agents alike, the theory would be to create an enterprise-ready, end-to-end solution, that autonomously correlated customer's specified grievances to self-service resources surfaced from a specialized LLM model.

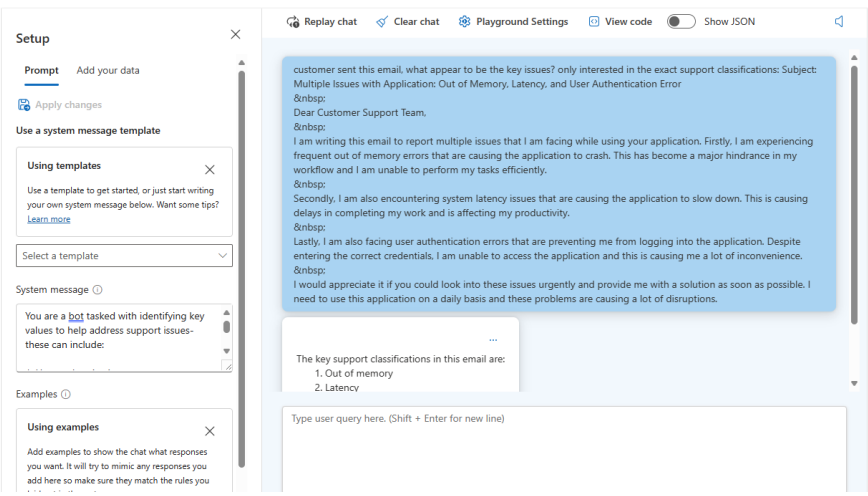

Figure 4. Azure OpenAI in practice

3.3 Generate "Badness"

Once error types and correlations are defined, and synthetic data is produced, we extrapolate badness for each error state. A 'badness' rate is assigned to each error state, which is a measure of severity or impact to a customer's operations. Each error state has a different penalty or cost associated with it, based on how severe the incident is and its effect on customers' operations. The model calculates the syndrome badness for each possible combination of bits, which equates to the total badness experienced by the customer when those individual bits cumulatively lead to a failure. For example, error state #7 in Figure 4 is the accrual of error bits 0, 1, and 2, represented as (1,1,1), and the syndrome badness is the sum of their individual badness rates, in addition to any badness caused by their interaction. To further emphasize the point, error state #7 includes all the error states 1 (001), 2(010), 3(011), 4(100), 5(101), 6(110), and 7(111), each containing its own badness. Therefore, the sum of the badness rates is the

syndrome badness, but the reported error badness indicates the syndrome that is most compelling, or consistent with our terminology, the 'baddest.'

	Error #	BPF	MFT	Badness/T	Ground Error Bits	Ground Error States	Syndrome	Syndrome Badness	Reported Error	Residual Badness	Pct Residual
0	0	0	0	0.000000	[0, 0, 0, 0, 0]	[]	[]	0.000000	0	0.000000	-
1	1	6	578	0.010381	[1, 0, 0, 0, 0]	[0]	[1]	0.010381	1	0.000000	0.0%
2	2	10	964	0.010373	[0, 1, 0, 0, 0]	[1]	[2]	0.010373	2	0.000000	0.0%
3	3	4	132	0.030303	[1, 1, 0, 0, 0]	[0, 1]	[1, 2, 3]	0.051057	3	0.000000	0.0%
4	4	9	715	0.012587	[0, 0, 1, 0, 0]	[2]	[4]	0.012587	4	0.000000	0.0%
5	5	3	18	0.166667	[1, 0, 1, 0, 0]	[0, 2]	[1, 4, 5]	0.189635	5	0.000000	0.0%
6	6	19	339	0.056047	[0, 1, 1, 0, 0]	[1, 2]	[2, 4, 6]	0.079008	6	0.000000	0.0%
7	7	20	618	0.032362	[1, 1, 1, 0, 0]	[0, 1, 2]	[1, 2, 4, 3, 5, 6, 7]	0.318721	5	0.010373	3.3%
8	8	3	78	0.038462	[0, 0, 0, 1, 0]	[3]	[8]	0.038462	8	0.000000	0.0%
9	9	20	421	0.047506	[1, 0, 0, 1, 0]	[0, 3]	[1, 8, 9]	0.096348	9	0.000000	0.0%
10	10	6	49	0.122449	[0, 1, 0, 1, 0]	[1, 3]	[2, 8, 10]	0.171284	10	0.000000	0.0%
11	11	15	61	0.245902	[1, 1, 0, 1, 0]	[0, 1, 3]	[1, 2, 8, 3, 9, 10, 11]	0.505375	11	0.000000	0.0%
12	12	10	950	0.010526	[0, 0, 1, 1, 0]	[2, 3]	[4, 8, 12]	0.061575	8	0.012587	20.4%
13	13	14	21	0.666667	[1, 0, 1, 1, 0]	[0, 2, 3]	[1, 4, 8, 5, 9, 12, 13]	0.952795	13	0.000000	0.0%
14	14	6	293	0.020478	[0, 1, 1, 1, 0]	[1, 2, 3]	[2, 4, 8, 6, 10, 12, 14]	0.270923	10	0.012587	4.6%
15	15	11	20	0.550000	[1, 1, 1, 1, 0]	[0, 1, 2, 3]	[1, 2, 4, 8, 3, 5, 9, 6, 10, 12, 7, 11, 13, 14...	2.020710	13	0.010373	0.5%
16	16	11	383	0.028721	[0, 0, 0, 0, 1]	[4]	[16]	0.028721	16	0.000000	0.0%
17	17	10	411	0.024331	[1, 0, 0, 0, 1]	[0, 4]	[1, 16, 17]	0.063432	16	0.010381	16.4%
18	18	13	315	0.041270	[0, 1, 0, 0, 1]	[1, 4]	[2, 16, 18]	0.080364	18	0.000000	0.0%
19	19	10	557	0.017953	[1, 1, 0, 0, 1]	[0, 1, 4]	[1, 2, 16, 3, 17, 18, 19]	0.163332	18	0.010381	6.4%
20	20	20	162	0.123457	[0, 0, 1, 0, 1]	[2, 4]	[4, 16, 20]	0.164765	20	0.000000	0.0%
21	21	7	82	0.085366	[1, 0, 1, 0, 1]	[0, 2, 4]	[1, 4, 16, 5, 17, 20, 21]	0.451509	5	0.028721	6.4%
22	22	12	290	0.041379	[0, 1, 1, 0, 1]	[1, 2, 4]	[2, 4, 16, 6, 18, 20, 22]	0.313835	20	0.010373	3.3%
23	23	18	66	0.272727	[1, 1, 1, 0, 1]	[0, 1, 2, 4]	[1, 2, 4, 16, 3, 5, 17, 6, 18, 20, 7, 19, 21, ...	0.953925	23	0.000000	0.0%
24	24	13	418	0.031100	[0, 0, 0, 1, 1]	[3, 4]	[8, 16, 24]	0.098283	8	0.028721	29.2%
25	25	4	820	0.004878	[1, 0, 0, 1, 1]	[0, 3, 4]	[1, 8, 16, 9, 17, 24, 25]	0.185378	9	0.028721	15.5%
26	26	20	366	0.054645	[0, 1, 0, 1, 1]	[1, 3, 4]	[2, 8, 16, 10, 18, 24, 26]	0.327020	10	0.028721	8.8%
27	27	9	851	0.010576	[1, 1, 0, 1, 1]	[0, 1, 3, 4]	[1, 2, 8, 16, 3, 9, 17, 10, 18, 24, 11, 19, 25...	0.718849	11	0.028721	4.0%
28	28	2	262	0.007634	[0, 0, 1, 1, 1]	[2, 3, 4]	[4, 8, 16, 12, 20, 24, 28]	0.252487	20	0.038462	15.2%
29	29	1	830	0.001205	[1, 0, 1, 1, 1]	[0, 2, 3, 4]	[1, 4, 8, 16, 5, 9, 17, 12, 20, 24, 13, 21, 25...	1.259486	13	0.028721	2.3%
30	30	5	338	0.014793	[0, 1, 1, 1, 1]	[1, 2, 3, 4]	[2, 4, 8, 16, 6, 10, 18, 12, 20, 24, 14, 22, 2...	0.613921	20	0.171284	27.9%
31	31	11	83	0.132530	[1, 1, 1, 1, 1]	[0, 1, 2, 3, 4]	[1, 2, 4, 8, 16, 3, 5, 9, 17, 6, 10, 18, 12, 2...	2.913274	13	0.080364	2.8%

Figure 4. Summary of Error States and Badness

3.4 Generate Actions

The next step in our process is to generate deterministic actions for the Q-learner to apply towards the learning process. Originally, under the auspices of a single resolution path, exploiting human capital to resolve syndromes, the action space was 31. This was based on each error bit having a binary outcome, 0 or 1,

henceforth, 2^5= 32, and since error bit, 0, was a non-event, 31 was the result. Incorporating an added action instrument where either a LLM-invoked self-service approach is followed, support agents swarm a complex error scenario, or potentially take place, the size of the action space increases to 243-1 (for the non-event) ending at 242 actions. Distilled down this looks like:

(5) 1-bit error states * 2^1 actions= 10
(10) 2-bit error states * 2^2 actions = 40
(10) 3-bit error states * 2^3 actions= 80
(5) 4-bit error states * 2^4 actions= 80
(1) 5-bit error states * 2^5= 32

242 represents all combinations of refer-to-GPT and refer-to-human-group. Using the previous error code 7 example, encoded as (1, 1, 1), there are 2^3=8 ways to handle the error state: HHH, HHG HGH HGG GHH GHG GGH GGG, where H= human and G=GPT. Hence, the action will be an assignment of each reported error bit to either a human or a GPT solver, or both. Finally, it is worth noting that no more than 5 agents will ever be tasked as our policy is dictated by 5 error bits.

3.5 Resolution and Reward Status

The resolution and reward step transpires after actions are generated with the objective to apply the best support policy for each error state. The Q-learner statuses of each support syndrome are attained by employing an epsilon-greedy policy. This means we are “greedy” 1-epsilon percent of the time and random epsilon percent of the time, with the end goal of learning and achieving the most optimal policy which prompt actions that lead to a resolution. By balancing exploration and exploitation, the learner may choose a random action to explore new resolution possibilities, while in some scenarios choosing the best action according to the Q-table. The model uses a discount factor to weigh immediate rewards heavily over future rewards. This encapsulates the greedy dynamic wherein the learner seeks to resolve a syndrome as quickly and efficiently as possible. Over time, epsilon shrinks to almost 0, as part of the training that occurs, so that everything becomes optimal. The solvers return either a successful or an unsuccessful solution per bit, with success probabilities of 0.98 for humans and 0.7 for LLM. A learning rate is also enacted to control the amount of new information updated to the Q-table. The model compares the true error state with the support policy action vector, computing the total reward or cost of the chosen actions. Rewards and costs are the derivative of several factors, including the type and number of errors, the accuracy and cost of human and LLM involvement, the correlation and noise of the errors, and the overall absence of customer dissatisfaction (not to be confused with customer satisfaction).

3.6 Q-Learning Recursion

Given Q-learning is a reinforcement learning algorithm, recursion happens based on the learning that occurs until the best possible outcome is achieved. For this study, the best outcome is to minimize cost for each error state, given the existential parameters inputted in the model. In addition to the parameters already mentioned, factors including residual badness and reopened tickets all influence the actions the model takes. Hence, the Q-table is updated according to good outcomes that are reached, thereby appending the states, and the states immediately preceding. Also, the Q-table is updated to reflect the new reward or cost values and improves its policy over time.

Three meta parameters are invoked, including Epsilon, which was previously introduced and distilled. Gamma is a discount factor that "discounts" future reward to obtain its effect on the updated value of the current state-action. Lastly, there is alpha, which reports how much of the step toward the new value we are willing to take. For example, consider:

alpha=1: 'all of it'
alpha=0.5: 'half of it'
gamma= .4

Provided a new state-best-action value, wherein a value is -20, then .4*-20=-8, which would accrue for the previous state. If the previous state has a value of -50 and the reward for the action is -30, then -38 would be the new value under the assumption alpha= 1. Alternatively, if we only take half of the step (alpha=0.5), then the new value of the -50 state-action is not -38 but -44. In essence, alpha represents the learning rate. When deciding to apply a solution to, for example, error bits 0,2,3, which correspond to 1,4,8=13, we have 8 ways we can achieve an outcome. Each outcome has a different internal reward meaning there is a different chance for success, and therefore has its own total reward.

4 Results

The objective of this research was to determine, given, partially observed data or an absence of data, altogether, whether a model could be developed that learns optimal customer support strategies. In terms of hypotheses:

H_0 = There is no evidence to suggest that a model can be developed to learn optimal customer support strategies given partially observed data or an absence of data

$H_a \neq$ There is evidence to suggest that a model can be developed to learn optimal customer support strategies given partially observed data or an absence of data

Developing, a Q-Learning Human-Only Solver, followed by a Q-Learning Hybrid Human-LLM Solver, adhering to the Markov Decision Process framework, our results demonstrate there is sufficient evidence to suggest both models can successfully learn optimal customer support strategies, therefore, supporting the alternative hypothesis over the null hypothesis. In Figure 5, there is a positive linear correlation between experience gained over time by the learner and the negative rewards realized in relation to the error states. Error state 31, an agglomeration of all possible symptoms derived from all 5 error bits, initially has the highest negative reward as the learner draws from its experience to produce the most optimal resolution path.

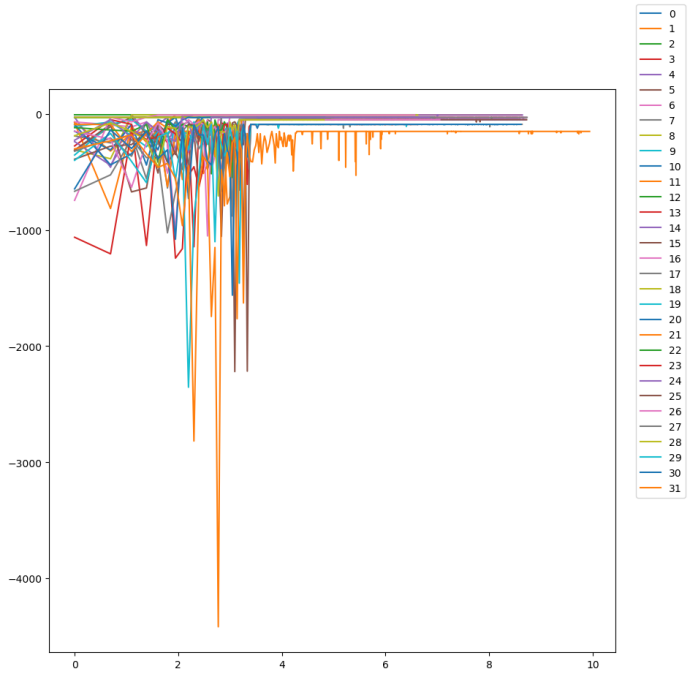

Figure 5. Q-Learning Human-Only Solver Results

The next plot depicts the epsilon-greedy strategy occurring over time, and the way the model learns over time. Epsilon greedy, a parameter embedded for, both, human only and hybrid-solvers, starts with an epsilon=

1. This denotes the model exclusively executing random, defined actions. The x-axis represents n-error messages, and as the Q-learner resolves incoming incidents through available action-states, epsilon greedy experiences a precipitous decay. Notably, after 20,000 messages, the learner moves from almost entirely randomized execution of available actions to about a 50/50 decision. After 100,000 incident examples, the learner trusts its learned policy 95% of the time, but is still willing to explore or experiment 1 time for every 20 instances. In essence, the Q-learner becomes more prescriptive in resolving support incidents.

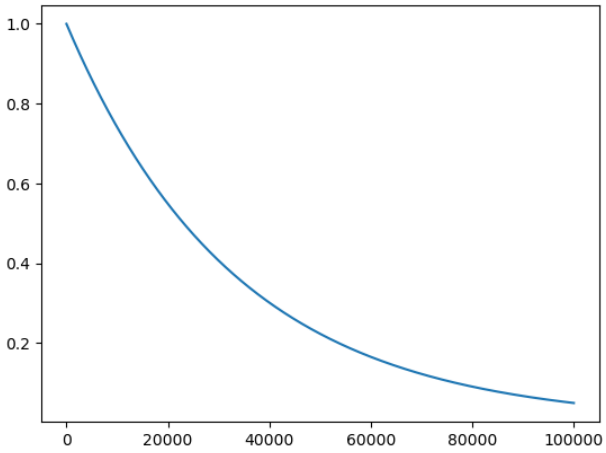

Figure 6. Epsilon-greedy strategy

The following set of plots illustrate the hybrid human-LLM solver in practice. For clarity and granularity, the plots have been separated by error states associated with error bits, 1-5, with the last plot, inclusive of 4 and 5-bit error states, as there is only (1) 5-bit error state, 31. The one major distinction between the two models, given the addition of the LLM-resolution, is the hybrid model has 242 state-action pairs available whereas the human-solver had only 31.

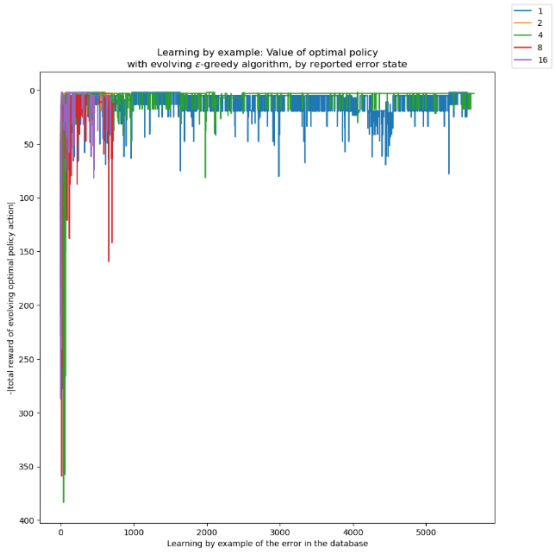

Figure 7. Q-Learning Hybrid Human-LLM Solver 1-bit Error State

Figure 7 offers clear evidence of learning. The initial exploratory phase involves many costly full or partial-tool-belt response scenarios. For instance, the user may be reporting error state 8, a one-bit error, but the agent responds with an array of resolutions that may or may not be related to the one actual error bit fault, in question. Over time, experience may teach us the best fix is to fix the 3rd error bit, which is the error bit associated with error state 8. In similar manner, we may learn to fix error state 10 (8+2, or error bits 3 and 1), because these bits might co-occur often enough to make the double-fix the best average strategy. In this scenario, error bit 1 is "unobserved", effectively a non-event. It is for this reason, we use Q-Learning to solve the POMDP problem, rather than some other MDP strategy that assumes completely-observed states.

Intuitively, it makes sense for the 1-bit error to quickly learn the best action to take given the linear nature of a support incident comprised of a single error. Before 1000 observed support incidents the Q-learner applies the optimal policy with negative rewards (or the cost) never exceeding 100. In Figure 8, while a logarithmic visualization is displayed, a similar correlation between time/experience and diminishing negative reward is observed. The reasoning behind the learner not consistently trending upwards is due to the learner determining some of the best scenarios are ones that produce the lowest cost or negative reward- this entails the LLMs being leveraged as an action to resolve a ticket, which means that action is inclined to fail sporadically. There is also a correlation between the order of magnitude of support tickets the learner solves for and an increase in LLM-led resolution failures.

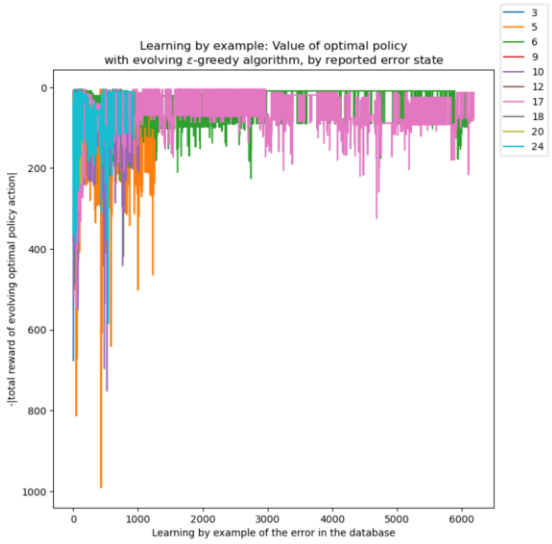

Figure 8. Q-Learning Hybrid Human-LLM Solver 2-bit Error States

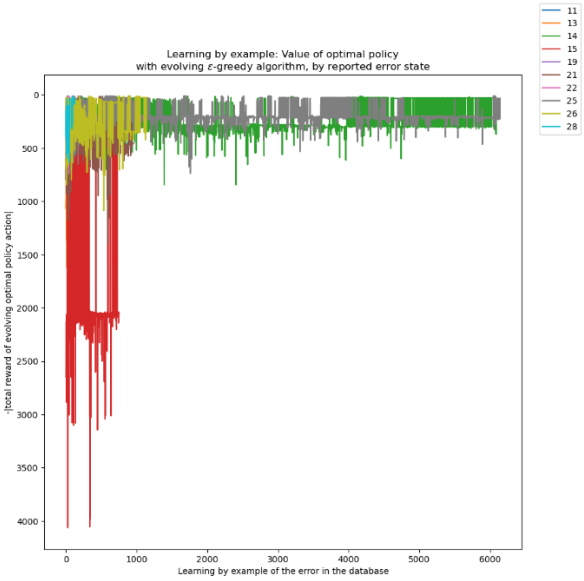

Figure 9. Q-Learning Hybrid Human-LLM Solver 3-bit Error States

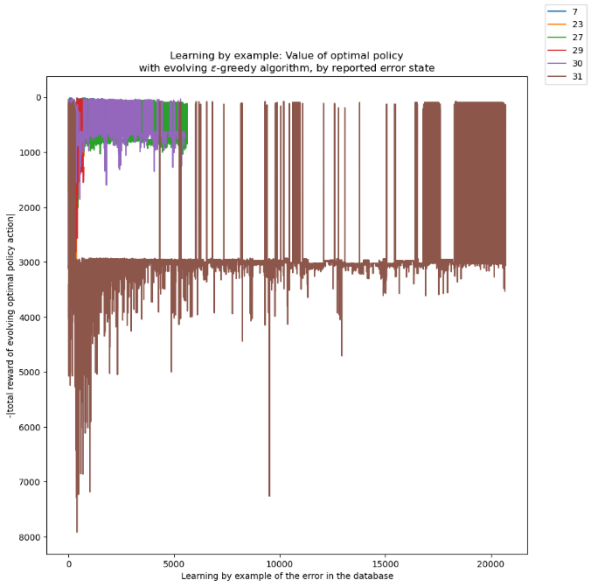

Figure 10. Q-Learning Hybrid Human-LLM Solver 4 and 5-bit Error States

Figures 9 and 10 illustrate the remaining state-action sets and how Q-learning performs against them. In Figure 10, you will notice error state 31 (brown) is the most variable amongst all error states. Due to there being more error states to resolve, the probability of LLM-solution failure increases, and therefore, the most likely outcome is "failure to solve" provided the relatively harsh penalty. On the other hand, occasionally the LLM-enabled resolution performs spectacularly. This explains the periodic high reward outcomes but the predominant low reward outcomes, even when the optimal policy is employed most of the time.

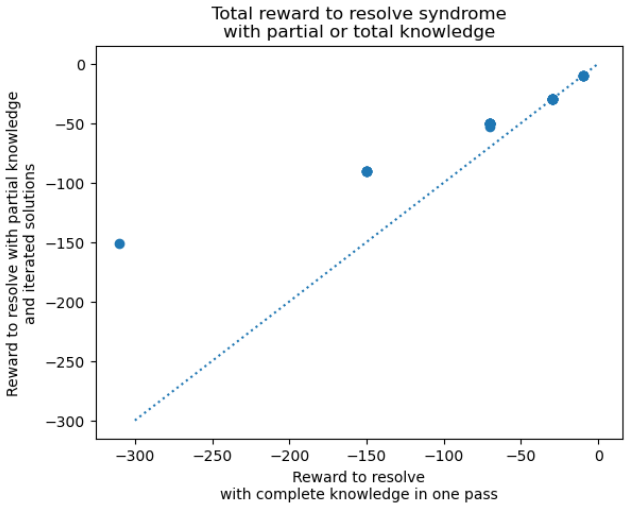

Figure 11. Q-Learning Human-Only Solver

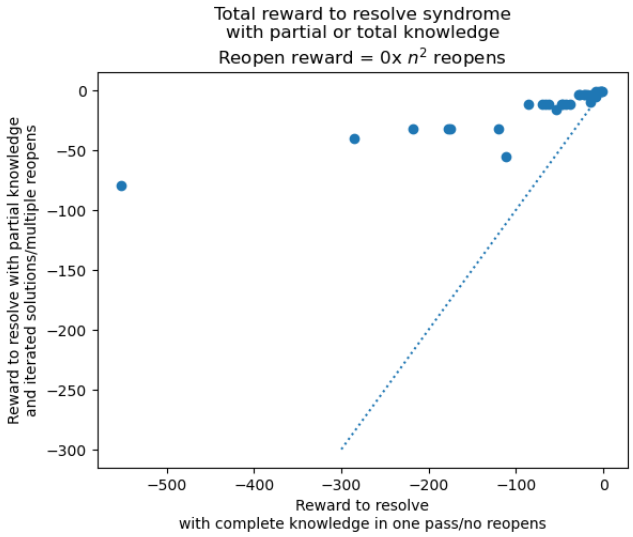

Figure 12. Q-Learning Hybrid Human-LLM Only Solver

Lastly, Figures 11 and 12 represent the human-only and hybrid models, respectively. The dots above the line signify error bits where the learner determined it was

cheaper to solve the problem with multiple support ticket reopenings versus resolving the problem through the available action states. Considering this, it is worth highlighting that this simulation is parametric, meaning arbitrary values were placed on rewards, for example. Hence, the negative reward associated with a ticket reopening could be modified, along with other observed parameters, potentially changing the overall dynamic of the learner, including executed policies, and customer support processes.

5 Discussion

Our research consisted of engineering a framework and models based on partially observed details of a customer support problem statement. Through this process we explored and exploited various state-action scenarios to realize the most optimal path to resolution, based on arbitrary cost considerations. Given this study was a simulation, with benchmark data, the work conducted can be leveraged to demonstrate how our models would work when parameters are adjusted, and enterprise-support specific features and observations were inputted. Moreover, as part of the bifurcation of support actions, human or LLM-based, separate modeling could be conducted respective to the LLM, including enforcing prompt engineering techniques which were expounded upon earlier in the study. The performance of the LLM, based on applied prompt engineering techniques, could further inform us how viable the self-service scenario would be and to what extent in a support situation, especially those more complex, in nature.

Also, an economically pessimistic point of view was instituted in our model development approach. For instance, the assumption was made that users cannot be trusted in support ticket entries- in other words, our model assumes support ticket entries are obfuscated in some way. While anecdotally this aligns to real world happenings, quantifying the abstraction was done arbitrarily. Also, an assumption is made that internal collaboration across product and engineering groups is ineffective and convoluted- this, again, is intuitively consistent with real life observations. Organizations, like Microsoft, have complex, highly integrated offerings which can eventually lead to complex support scenarios. There is also the assumption LLMs fail regularly, hence, the self-service path may not be optimal, especially in complex scenarios as we witnessed in the plots above. Finally, error bits were set to 50% marginal error rate, thus, given the parametric basis of the study, adjustments to the error rate could also impact the overall results of the study.

It is, perhaps, painfully obvious how rapid the technology landscape, and as a byproduct, AI, is evolving. Through this study, tracking and evaluating how

LLM's were being used in the real world, including continuous iterations and developments around prompt engineering, was increasingly difficult to follow. These advancements will surely continue, potentially given further credence to parts of this study or effectively supplanting approaches, like few-shot learning and chain of thought approaches to retrieval augmented generation.

That leaves, arguably, the most critical aspect of the discussion, ethics. First, as noted from at least one source, including, "Impact off artificial intelligence on employees working in industry 4.0 led organizations," (Malik et. al, 2021), evidence demonstrates the adverse effects of enterprise AI adoption. These included, but were not limited to, job insecurities, technostress, work overload, and added complexity. Moreover, as organizations aspire to drive operational efficiencies through the adoption of AI, these investments are always based on a business case supported through a return-on-investment (ROI) basis. This can and has been shown to mean, by employing X, we can save on Y, which will net a Z operational margin increase. The implementation of AI products in organizations, including Chegg and Stack Overflow, and the resulting reduction in workforce, has already substantiated this strategy, (Constantz, 2024).

The discussion around operational efficiencies naturally lead back to the structure of this simulation and the opportunity to translate this research to an empirical study. Organizations seeking ways to improve their customer-facing operations, including support, will inevitably weigh how process improvements and serving the needs of their customers impact their bottom lines. Commercial businesses operate on profitability and growth- hence, logically it makes sense that businesses resolve incidents and address customer concerns only insofar as they can remain profitable, even if that entails a less than suboptimal outcome for the customer.

AI development, from foundation models, to how they are modified to serve bespoke needs, and made available to end users and customers, requires responsible stewardship from everyone involved. This includes technology providers like Microsoft and OpenAI, and many AI developers cropping up, extending generalist foundation models with their own value propositions. This means being thoughtful of how bias may be introduced into a model, including when providing instructions to an LLM, effectively curating the prompt-completion process. In parallel, organizations need to consider the broad implications of adopting AI and its consequences on people's livelihoods including the downstream effects of displacement, at a microeconomic and macroeconomic level. Good stewardship, beyond how AI is developed and used, also entails supporting individuals through this historical transition, including education and

new work opportunities, where AI and humans can complement one another in advancing humanity.

6 Conclusion

Customer experience is a vital part of every business' strategy, one that can affect market share, based on customer retention, new client acquisition, and correlated overall revenue performance. In an enterprise's purchase lifecycle, how organizations support their customers, to realize the value of their purchase or investment, influences overall satisfaction and the likelihood of repeat business.

Customer support operations start with frontline agents, which traditionally has meant human agents interacting with customers to triage and resolve their support inquiries and concerns. With the advent of AI and more recently generative AI, enterprises are increasingly relying on these capabilities to augment their workforces or altogether replace them. This growing trend in consideration set the stage for our study, developing a simulation to identify the most optimal path to customer support incident remediation.

Our simulation invoked Markov Decision Process and a reinforcement learning technique called Q-Learning, to model a human-only and hybrid (human + AI) approach to resolving customer issues. Given an absence of data, data was contrived and represented as error bits, both as standalone entities and correlated symptoms, and processed through the Q-Learning algorithm to land on a resolution path balancing exploring and exploiting permutations of human and AI actions. In conclusion, despite a lack of data, we were able to learn a good strategy for approaching customer support that achieves a balanced resolution outcome. Moreover, as a parametric study, this infrastructure is malleable enough, provided changes to parameters, for example, increasing the negative cost associated with an unresolved ticket, to produce a more realistic result, respective to the enterprise and its customers.

Acknowledgments. We would like to thank Dale Sayers and Thad Schwebke for their input and contributions throughout this research process. We would also like to thank Dr. Jacquelyn Cheun-Jensen for her feedback throughout the capstone project., The authors also wish to acknowledge valuable conversations with Peter Leopold, Ph.D. and an algorithm for generating correlated binary synthetic data from Ilia Rushkin, Ph.D.

References

1. Cornfield, G. (2021, May). Recognizing Your Customer's Purpose is Key to Growth. Harvard Business Review. https://hbr.org/2021/05/whats-your-customers-purpose

2. Bamberger, S., Clark, N., Ramachandran, S., & Sokolova, V. (2023, July 06). How Generative AI Is Already Transforming Customer Service. Boston Consulting Group. https://www.bcg.com/publications/2023/how-generative-ai-transforms-customer-service

3. Ramachandran, S., Clark, N., & Sokolova, V. (2020, May 28). Redefining Customer Service for the Future. BCG. https://www.bcg.com/publications/2020/redefining-customer-service-for-the-future

4. Morgan, B. (2023, August 16). What Impact Will AI Have on Customer Service. https://www.forbes.com/sites/blakemorgan/2023/08/16/what-impact-will-ai-have-on-customer-service/?sh=1377b4bd6aa6

5. Ghosh, B., Prasad, R., Pallail, G., (2021). The Automation Advantage. McGraw-Hill. https://www2.deloitte.com/us/en/insights/focus/cio-insider-business-insights/reimagining-the-technology-operating-model.html

6. Bhatia, A. (2023, November 30). An Analysis of "The Power of Prompting" paper. https://www.linkedin.com/pulse/analysis-power-prompting-paper-ashish-bhatia-5u51f/

7. Cohen, H. (2023, July 05). Customer Self-Service: How Support, IT, and HR Can Do More with Less. Tango. https://www.tango.us/blog/customer-self-service

8. Gupta, D. (2022, December 22). Customer Self-Service: The Self-Help Support Model (2023). Whatfix. https://whatfix.com/blog/customer-self-service/

9. Pham, D.T., Chan, L.L, Alam, S., Koelle, R. (2021). Real-time departure slotting in mixed-mode operations using deep reinforcement learning: a case study of Zurich airport. Fourteenth USA/Europe Air Traffic Management Research and Development Seminar.

10. Awan, A. (2022). An Introduction to Q-Learning: A Tutorial for Beginners. DataCamp (2024). An Introduction to Q-Learning: A Tutorial For Beginners | DataCamp

11. Hahsler, M., Kamalzadeh, H., (2021, May). POMDP: Introduction to Partially Observable Markov Decisiono Processes. POMDP: Introduction to Partially Observable Markov Decision Processes (r-project.org). r-project.org

12. Berger, E. (2023, June 9). Grounding LLMs. Grounding LLMs - Microsoft Community Hub

13. Giacaglia, G. (2019, March 10). How Transfomers Work: The Neural Network used by OpenAI and DeepMind. Towards Data Science. https://towardsdatascience.com/transformers-141e32e69591

14. Pieck, C. (2012, Fall). Markov Decisions. Stanford (2013). CS221 (stanford.edu)

15. Baeldung. (2023, March 2024). Epsilon-Greedy Q-learning. Baeldung.

16. Nori, H., Lee, Y.T., Zhang, S., Carignan, D., Edgar, R., Fusi, N., King, N., Larson, J., Li, Y., Liu, W., Luo, R., McKinney, S.M., Ness, R.O., Poon, H., Qin, T., Usuyama, N., White, C., & Horvitz, E. (2023, November). Can Generalist Foundation Models Outcompete Special-Purpose Tuning? Case Study in Medicine. Microsoft.

17. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Sandhini, A., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandish, S., Radford, A., Sutskever, I., Amoedei, D. (2020, July). Language Models are Few-Shot Learners. OpenAI.

18. Eriksson, T. (2020, February 16). Think with me, or think for me? On the future role of artificial intelligence in marketing strategy formulation (2019). Emerald Insight.

19. Malik, N., Tripathi, S.N., Kar, A.K,, Gupta, S. (2021). Impact of artificial intelligence on employees working in industry 4.0 led organizations. Jaipuria Institute of Management, Lucknow, India, Department of Management Studies, Indian Institute of Technology Delhi, New Delhi, India, and Department of Information Systems, Supply Chain and Decision Making, NEOMA Business School, Reims, France. https://whatfix.com/blog/customer-self-service/

20. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., Polosukhin, I. (2017). Attention is All You Need. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

21. Collis, J. (2017, April 19). Glossary of Deep Learning: Word Embedding. Medium.com. medium.com/deeper-learning/glossary-of-deep-learning-word-embedding

22. Porter, S. (2023, August 28). Understanding Cosine Similarity and Word Embeddings. Medium.com. Understanding Cosine Similarity and Word Embeddings | by Spencer Porter | Medium

23. Chunjie, L., Jianfeng, Z., Leig, W., Qiang, Y. (2017, October 23). Cosine Normalization: Using Cosine Similarity Instead of Dot Product in Neural Networks. Arxiv.org

24. What is Azure OpenAI Service? (2024, February 15). Microsoft. https://learn.microsoft.com/en-us/azure/ai-services/openai/overview).

25. Steen, H. (2023, November 20). Retrieval Augmented Generation (RAG) in Azure AI Search. Microsoft. https://learn.microsoft.com/en-us/azure/search/retrieval-augmented-generation-overview

26. Bukowski, D. (2023, September 27). Context is Everything: The Importance of High-Quality Grounding Data. Medium.com. https://medium.com/@bukowski.daniel/context-is-everything-part-1-the-importance-of-high-quality-grounding-data-7a93dbaded96

27. Ani. (2023, December 2018). RAG – Retrieval-Augmented Generation. Medium.com. RAG — Retrieval-Augmented Generation | by Ani | Medium

28. Sanchez, A.G. (2024). Azure OpenAI for Cloud Native Applications: Designing, Planning, and Implementing Generative AI Solutions. O'Reilly.

29. Constantz, J. (2024, February 8). AI Is Driving More Layoffs Than Companies Want to Admit. Bloomberg. https://www.bloomberg.com/news/articles/2024-02-08/ai-is-driving-more-layoffs-than-companies-want-to-admit?embedded-checkout=true