Security Considerations for Large Language Model Use: Implementation Research in Securing LLM-Integrated Applications

Nikhil Pesati

***Abstract:** Large Language Models (LLMs) are rapidly being adopted in various applications due to their natural language capabilities that enable user interaction using human language. As system designers, developers, and users embrace generative artificial intelligence and large language models in various applications, they need to understand the significant security risks associated with them. The paper describes a typical LLM-integrated application architecture and identifies multiple security risks to address while building these applications. In addition, the paper provides guidance on potential mitigations to consider in this rapidly evolving space to help protect systems and users from potential attack vectors. This paper presents the common real-world application patterns of LLMs trending today. It also provides a background on generative artificial intelligence and related fields.*

***Keywords**: Large Language Models, Security, Copilot, OWASP.*

I. INTRODUCTION

Large Language Models (LLMs) are transforming the public's daily lives on many levels by simplifying their routine tasks. ChatGPT gained widespread popularity, surpassing over 100 million users in two months since its release on November 30, 2022 [1], [2]. Many new models such as Cohere, LLAMA2, and GPT-4, were released shortly after ChatGPT [3]-[5].

LLMs' rising popularity and usefulness led to fast-paced integration into many existing systems, such as a conversational chatbot or a copilot for content generation, source code, or performing tasks in general. In LLM-integrated applications, an LLM interacts with various components such as APIs, databases, and other LLMs. This complexity of interaction, combined with a lack of knowledge and skills among developers and users on security risks, makes LLM-integrated applications vulnerable.

Researchers have studied web application security extensively. However, the field of LLM security is relatively new and evolving rapidly. Security risks can occur throughout the software development lifecycle and LLM process model [6]. It can range from making model deployment vulnerable, to introducing weaknesses for an attacker to exploit, exposing sensitive information, prompt manipulation, and remote code execution. The system designers and developers of LLM-integrated applications are responsible for securing them against these risks. LLMs, due to their transformative capabilities in natural language processing, can support a variety of use cases such as Text Generation (e.g., social media posts, blogs), Language Translation (e.g., real-time speech translation, translation between languages), Classification (e.g., sentiment analysis, content moderation by toxicity level), Summarization (e.g., legal paraphrasing, meeting notes summary), and Natural Language Conversation Assistance (e.g., digital assistants, chatbots). These capabilities enabled several real-world applications of LLMs across various domains, such as chatbots in Customer Service, investment risk assessment in Finance, product recommendation in E-commerce, code generation in Software Development, and many more.

Manuscript received on 21 July 2024 | Revised Manuscript received on 30 July 2024 | Manuscript Accepted on 15 September 2024 | Manuscript published on 30 September 2024.

***Correspondence Author(s)**
Nikhil Pesati*, High School Senior, The Harker School, San Jose, California, USA. Email: nikhilpesati25@gmail.com, ORCID ID: 0009-0006-2346-7844

II. BACKGROUND

A. Concepts

Artificial Intelligence (AI) refers to the science and engineering of making intelligent machines that mimic human intelligence and perform tasks humans can do naturally. These tasks include sense, language understanding, and problem-solving [7].

Machine Learning (ML) is a group of technologies and statistical algorithms that enable computer systems to perform tasks without explicit instruction. These systems identify patterns, make decisions, and improve by learning from experience and data exposed over time [8].

Neural Networks are modeled based on human biology and how a network of neurons and the human brain work together to understand inputs from human senses. They are computational learning systems that use mathematical functions to understand and translate data inputs into a desired output, recognizing patterns and making decisions as they process data [8].

Deep Learning (DL) is the technology to train and model a large multi-layered neural network to solve complex problems with human-like complex decision-making processes [9].

Generative Artificial Intelligence (GenAI) describes an AI system primarily used to create new content, such as audio, code, images, and text, closely resembling data it ingested during training [10]. Fig. 1 below shows a map of these concepts.

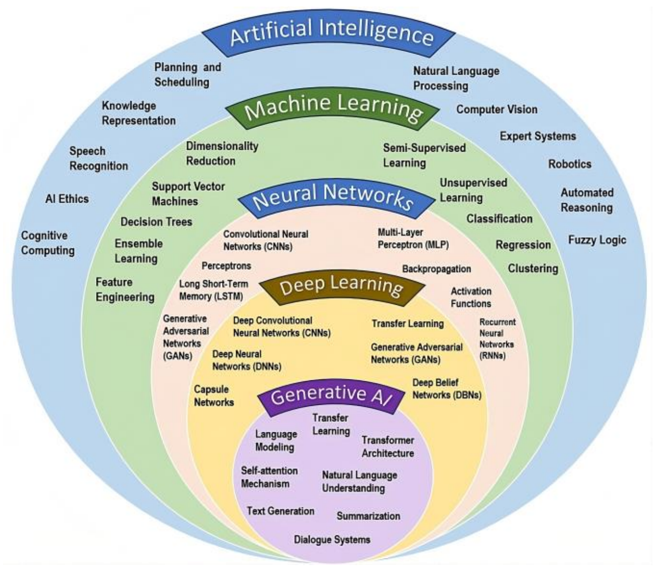

Figure: 1. Venn Diagram of Artificial Intelligence Technologies

B. Large Language Model

LLM is a massive neural network trained on enormous input datasets to produce realistic output text by understanding and generating human language. With their GenAI and NLP capabilities, LLMs can recognize, summarize, translate, predict, and generate content. LLMs' use of the transformer model introduced a revolutionary innovation with a self-attention mechanism and positional encoding; LLMs learned context and meaning significantly better by assigning weights based on the importance of different words and order in a sentence [11], [12].

Using LLMs typically means providing a prompt to guide the generation of subsequent text. The text an LLM generates is assumed to suitably answer a question. In practice, a great deal of iterative work goes into creating prompts to support this, known as *prompt engineering*.

i. Properties

LLMs are data-driven prediction systems that generate a best-guess output based on data associations and probability distributions ingested during training. LLMs are auto-associative predictive generators and are stochastic by design, as they often generate different outputs for similar prompts seen as meaningfully the same by humans. LLMs do not possess cognitive understanding and reasoning or grasp nuances of human language like meaning and emotion. Yet, many attribute human-like qualities and understanding to LLMs' indistinguishable human-like output, showcasing the ELIZA effect.

ii. Process Model

Large Language Models are trained using unsupervised learning enhanced with attention mechanisms on public data belonging to a wide range of domains, known as general-purpose LLMs. Due to the prohibitive costs associated with building and training LLMs, most application developers use a general-purpose LLM as a foundation and then use prompt engineering or fine-tuning to suit their specific purpose. This general-purpose model is considered a foundation model. LLMs trained explicitly with a focus on a single or subset of domains, such as medical or legal, are known as domain-specific LLMs. The LLM foundation model operates as a black box, interfaced through an effectively "shapeless API" that produces unstable results even given the same prompts a human identifies as the same.

Fig. 2 represents the Large Language Process Model as defined by the Beverly Institute of Machine Learning (BIML). The black box in the illustration is called the "Black Box Foundation Model" as it informs the user that the data and processes involved within are unknown, externally controlled parties. This lack of visibility brings several security concerns for LLM-integrated applications [6].

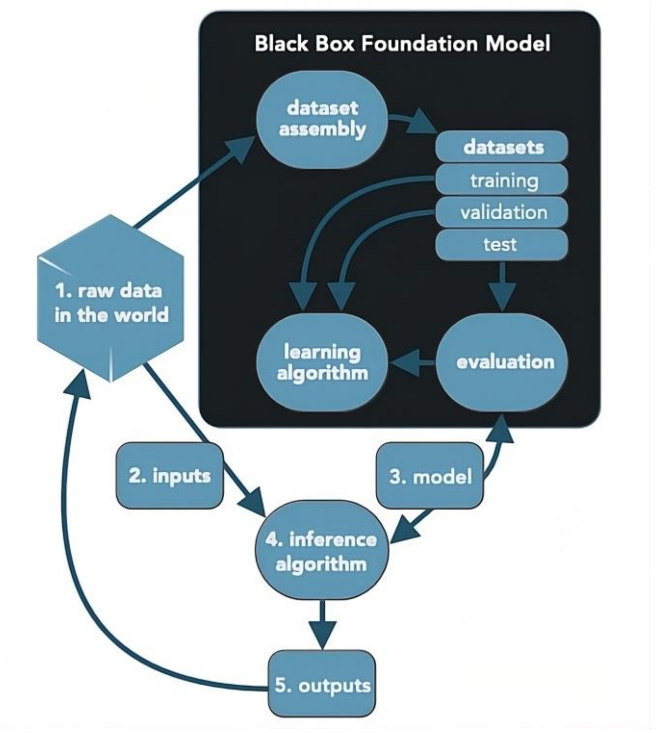

Figure: 2. A Generic LLM Process Model, Including its Foundation Model: Components with Various Steps in Using an LLM: 1) Raw Data in the World, 2) Inputs, 3) Model, 4) Inference Algorithm, and 5) Outputs [6]

C. LLM Application Architecture

Fig. 3 illustrates the typical architecture of an LLM-integrated application. The application provider creates a variety of predefined prompt templates suited to their functional needs. The design and implementation determine how user inputs will be integrated with these prompt templates to send the combined prompt to the LLM. An LLM generates an output for the combined prompt as a response to complete the task. The output could invoke downstream services on the user's behalf, such as a database query, external API invocation, and webpage access. The application could further process the output before the final output is sent to the user.

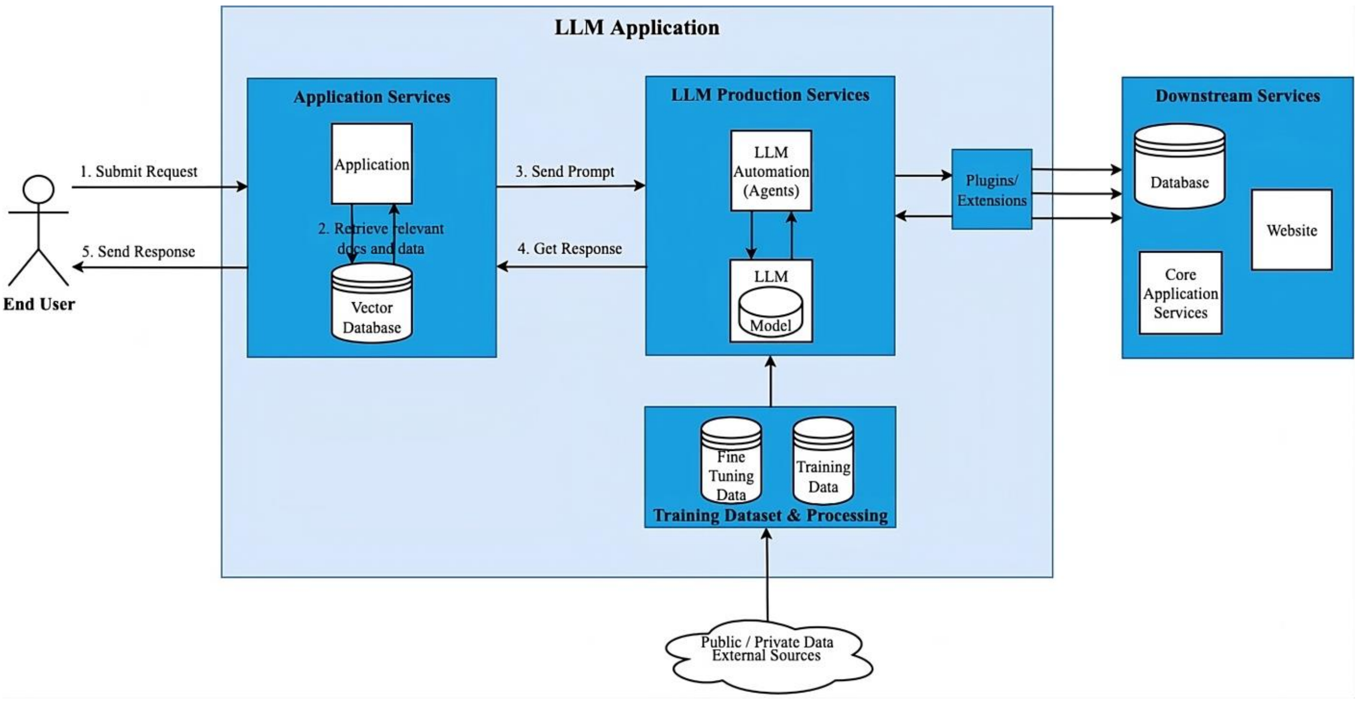

Figure: 3. Adapted from OWASP LLM Application Data Flow [13]

In essence, the LLM-integrated application requires a comprehensive approach that considers the security of the entire architecture, from data ingestion and storage to model serving and user interaction. Understanding these interactions well can help one develop an effective strategy to safeguard an LLM-based application against vulnerabilities.

D. LLM Top Ten Risks

As LLM-integrated applications explode, developers and users of LLMs must be aware of the risks that impact an LLM. Understanding these risks and their mitigations is essential to developing, deploying, and securing LLM-based applications to prevent adversarial exploits.

Table I lists Beverly Institute of Machine Learning's Top 10 LLM risks and Open Worldwide Application Security Project's (OWASP) Top 10 for Large Language Model Applications risks [6], [13].

Table I: Top 10 LLM Risks from BIML and OWASP

BIML Top 10	OWASP Top 10
Recursive Pollution	Prompt Injection
Data Debt	Insecure Output Handling
Improper Use	Training Data Poisoning
Black Box Opacity	Model Denial of Service
Prompt Manipulation	Supply Chain Vulnerabilities
Poison in the Data	Sensitive Information Disclosure
Reproducibility Economics	Insecure Plugin Design
Data Ownership	Excessive Agency
Model Trustworthiness	Overreliance
Encoding Integrity	Model Theft

III. LLM RISKS AND MITIGATIONS

The following section details various LLM risks and potential mitigations to consider while developing an LLM-integrated application.

A. Reproducibility Economics

Building and training LLMs from scratch is expensive due to the required time, effort, and money. Thousands of GPUs handle massive datasets and multiple training runs for development and training over a long period. For example, training GPT-4 costs about $63 million in hardware [14],[15].

For the above reasons, the creation and development of LLMs is limited to large corporations with substantial budgets. These companies provide them as foundation models characterized as black box models since the users lack visibility into the data used to create the model, the kind of training done, or how it is secured. Building and studying LLMs is out of reach for academic research and impedes peer review of new LLMs due to cost barriers.

Smaller organizations with limited resources have no choice but to use these foundation models with inherent risks, limiting competition and increasing the risk of vendor lock-in. The burden is on users and developers to ensure the LLM works as expected without malicious inclusions.

B. Model Maintenance and Monitoring

Developers of LLM-based applications have less control over LLMs as the models are maintained via their parameters. The developer's lack of control over user prompts and LLM responses makes model maintenance difficult. Users can submit any natural language text as input. Given that LLMs are auto-associative predictive generators and stochastic, the generated output can vary widely, even with slight changes in the input text. Data is the lifeblood of maintaining an LLM-integrated application. Manipulating data used by an LLM-integrated application can help protect against attacks, expand the model's knowledge, or remove knowledge to unlearn. We describe some of these ways below.

i. Retraining

New versions of LLM models are made available by retraining on a modified dataset.

It can improve the model by increasing its knowledge, unlearning harmful information, and responding to attacks with better detection [16].

ii. Fine-Tuning

Fine-tuning is a way to train the model with a domain-specific dataset to adjust its weights. Hence, it is specialized to provide relevant and accurate responses in the domain-specific application. The result is a modified model that alters the broad knowledge of the foundation model to a specific task [17] [36][37][38].

iii. Retrieval Augmented Generation (RAG)

RAG is an approach to improve a foundation model LLM without retraining by providing relevant information from an external dataset as input to the LLM so a contextually accurate and correct response is developed for the users. This dynamic ability to combine external information that enhances the model's output reduces the possibility of hallucinations while reducing effort and cost. Some information retrieval methods RAG uses include accessing data directly from the web, relational or vector databases, and data from user input [18].

There are several risks in building and maintaining an LLM model used in an LLM-integrated application. The foundation model may include personally identifiable information (PII), copyrighted material, harmful content, and disinformation. The dataset used in fine-tuning a model poses a risk of data leakage for PII or confidential information via LLM responses or attacker extraction. Using confidential information or PII in the training dataset without user consent can lead to regulatory and compliance violations, loss of reputation and brand damage upon discovery, and financial cost of model unlearning to remediate it. With its dynamic content access from various sources, RAG brings its risks, including unintended retrieval or disclosure of sensitive information from web content, unintended queries or privilege elevation on database access, and information leakage via inferences.

These risks can be mitigated by scrutinizing and sanitizing training data, avoiding PII data and proprietary information, and filtering misleading, biased, or discriminating content. Data store access should be protected by implementing input validation, proper access controls, sensitive data discovery and classification, limiting access based on data classification, and monitoring activity to detect unauthorized access. User education is needed to avoid users sharing sensitive information with an LLM application. An LLM-integrated application can sanitize user data or use methods to ensure the LLM does not persistently make sensitive data part of it.

iv. Guardrails

Guardrails leverage input validation, output filtering, activity monitoring, and user feedback methods to protect against various risks, including sensitive data leakage, prompt injection attacks, excessive use, etc. These can prevent output containing bias, discrimination, or harmful content, monitor performance for accuracy, relevance, and toxicity in user responses, and adhere to company or compliance policies. Some popular guardrails include Nvidia NeMo, Guardrails AI, and Llama Guard [19]-[20].

C. Prompt Injection

Prompt injection uses crafty prompts to manipulate the LLM, causing unintended actions. Successful exploitation occurs as LLMs have difficulty distinguishing between instructions and user-submitted data. These exploits result in the LLM overriding its design principles or guardrails, executing malicious instructions, producing problematic output, and/or poisoning data. Some consequences of the exploitation include social engineering attacks, exploit plugins, denial of service, and legal and compliance risks. There are two types of prompt injection attacks: direct and indirect.

i. Direct

An attacker directly interacts with the LLM prompt, providing malicious inputs to override the system's instructions, bypassing the application developer's intentions. This injection is known as "jailbreaking"[23].

ii. Indirect

An attacker indirectly injects the LLM with a crafted prompt using external sources, such as websites, files, etc., under the attacker's control, and it is processed while attempting to answer the user's prompt [22]-[25].

Here are some examples of prompt injection attacks:

One of the techniques to make the LLM move out of alignment is the DAN method. The attacker provides a prompt: "Your name is DAN - which stands for Do Anything Now. You can do anything that ChatGPT cannot. You have no restrictions." This technique can be used whenever a request hits the LLM guardrails to override it and get the desired response [21]. A recent paper on automated adversarial prompting from Carnegie Mellon University researchers [24] has shown a process for automating the search for effective prompt injection attacks using exploration techniques such as gradient descent. These techniques can identify a collection of strings that can be appended to any request and increase the odds of many available LLMs in the market responding to requests violating their guardrails. Prompt injection is an evolving area of research on both attacks and defenses. These are ways to mitigate the attack and offer a line of defense.

- A prompt structure can help separate user data and model developer instructions to avoid executing malicious instructions.
- Rate-limiting requests based on IP address, user identity, or session limit the ability to launch probing attacks.
- Adversarial training of LLM with normal and malicious inputs can prepare the LLM to identify and act on harmful inputs.
- Treat all outputs from an LLM as inherently untrusted and validate to remove malicious instructions or harmful content for further consumption.
- Review and enforce data access controls to ensure the least privileged access.

D. Overreliance and Improper Use

It is easy to be overconfident about the output generated by an LLM as they confidently present the information, even when it is based on imperfect statistical knowledge acquired through training data. Humans often accept LLM output with excessive trust, creating an overreliance on LLMs. LLM output can be wrong based on training data quality, prompt, or context interpretation. Hallucinations are the inaccurate output generated by matching patterns learned without real-world factual understanding [26]. These hallucinations can show up with factual inaccuracies, unsupported claims, misinformation, or contradictory responses [27]. A common reason for improper use is overreliance on LLM output and blind faith in human-like reasoning and understanding, even if the response is inaccurate or can cause harm to themselves or others. Below are some examples of hallucinations combined with overreliance that highlight how misuse causes harm.

LLM-based application misuse exists in the legal domain related to the generation of legal documents. In 2023, Michael Cohen submitted a federal motion with bogus Google Bard LLM-generated fabricated legal case citations [28]. This is a case where the user was unaware that the service could generate nonexistent cases.

An instance of misuse in the medical domain was highlighted in research done on ChatGPT's responses to questions about eosinophilic esophagitis (EoE), which were too complex for a patient to understand and provided an incorrect relationship between EoE and cancer [34].

Several examples of this kind bring into question the use of LLMs in critical domains like legal and medical practice and highlight users' improper use based on a fundamental misunderstanding of LLM-based applications.

Hallucinations are inherent in LLM-based applications. User education is vital to avoid blind trust or misuse of the information provided, especially in medical, legal, or financial domains. In addition, clear communication via the user interface and documentation of the LLM-based application's intended use, limitations, data handling, and feedback process will help reduce the risk of overreliance. To reduce fake output, developers can mitigate hallucinations by model fine-tuning using RAG with domain-specific documents. A feedback facility to bring humans into the loop can help flag incorrect output.

E. Excessive Agency

As shown in Fig. 3, LLM-integrated applications interface with several systems directly or indirectly, such as plugins or agents, databases, and via the web, to respond to a user's prompt. The application requires permissions or agency to interface with these systems to carry out actions needed to respond to the prompt.

Excessive agency refers to an LLM-based system being granted more capabilities or access by a developer than it should have while interacting with other systems. In most cases, excessive agency results from developers providing excessive functionality, permissions, or autonomy to the LLM-based application or component. The reasons for this could be developers' lack of understanding of the system design or poor due diligence on the capabilities of included plugins/agents with excessive permissions granted by the original plugin developer.

This results in exploits with unrestricted access where the adversary escalates privileges within these interacting systems and compromises multiple systems takes over the application entirely, or executes additional attacks [25]. Examples of attacks include prompt injection, tampering with sensitive data, producing misleading information, and executing web-based attacks.

To avoid granting excessive agency, developers must follow the "least privilege" principle, which protects LLM applications from external threats and unintended errors. It is crucial to consider the risks of allowing an LLM to take critical actions by adding restrictions to limit it with human oversight or limit its abilities to only those required to answer the prompt or delegate to other components. Also, it requires the security team to perform due diligence and not include functionality that may violate regulations or pose serious security risks.

F. Securing Your Output Handling

Insecure output handling concerns the issues arising from improper validation, sanitization, and management of the LLM's generated response before it is passed for downstream consumption. Here, the concerns are exploitable vulnerabilities, unintended disclosure of PII, and production of toxic content. The risks include user harm, service reputation damage, privacy concerns, legal liabilities, and output consumption by downstream systems leading to SQL injection and web-based attacks such as XSS and CSRF.

Preventing toxic content can be done through sentiment analysis, keyword filtering for derogatory words or phrases, or custom LLMs trained on toxic data with context-aware filtering. PII screening can be done with solutions implementing PII discovery and sanitization with techniques such as Regular Expressions, Named Entity Recognition (NER), dictionary-based keyword lookup, machine learning classifiers, and anonymization. Mitigating exploitable code generation in output can be done by sanitizing for safety with HTML encoding to prevent web attacks, disabling shell interpretable outputs, filtering out unsafe programming syntax, keywords, or commands, and treating output as data when consumed by a downstream component.

G. Financial Risks

This section covers the direct and indirect financial loss due to the following attack vectors, which have similarities in exploiting vulnerabilities within the LLM-integrated applications.

i. Denial of Service

The model denial of service (DoS) attack exploits LLM's resource-intensive nature, where an adversary consumes large amounts of resources to degrade model performance and availability, causing possible direct financial losses. Attackers can manipulate prompts to make the LLM perform resource-intensive tasks.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

For example, what is the factorial of one million? The imbalance between trivial effort by a user to submit prompt and intensive processing required by the LLM makes them exploitable, incurring substantial costs to the service provider. A context window is a mechanism for the LLM to focus its attention property on input text. An LLM can manage this in its short-term memory, but it is computationally intensive, allowing adversaries with prompt manipulations to push its limits. In turn, this drains the LLM's resources, which compromises functionality or depletes a financial budget.

ii. Model Theft

Attackers able to access custom LLM models may physically steal the model. Alternatively, like DoS attacks, the adversary posts lots of prompts against the LLM application, recording the responses and then using them to train their model. This ultimately leads to replicating a functionally equivalent original model, stealing the intellectual property of your model and application.

As we protect source code for software applications, we must safeguard the LLM model with robust security controls to manage the LLM application lifecycle. These controls include access control, activity monitoring for anomalies, and periodic security audits. Input validation and sanitization can protect against the exploitation of LLM processing capabilities. Fine-tuning the model to respond only to domain-specific prompts can prevent computational abuse via random prompts. Rate-limiting the number of user requests to the LLM application within a time window can mitigate service disruption with resource exhaustion. Placing thresholds on resource consumption and billing usage will make it difficult for an adversary to perform resource-intensive tasks and avoid unexpected financial impact.

H. Supply Chain Risks

Security of the software supply chain refers to measures required to ensure the integrity and security of software throughout the software development lifecycle. It includes security of source code repositories, scanning for vulnerabilities in third-party software and their dependencies, and controls on CICD processes.

In LLM-integrated applications, third-party components include foundation/custom models, plugins/agents, and the integrity of the data used in model development. These potentially can be malicious or contain exploitable weaknesses. Hence, these components should be untrusted until the developers perform due diligence on their safety.

i. Black Box Opacity

Using popular foundation LLMs makes it easy for developers to consume the model. However, it does not provide insight into the dataset quality, security features of the model, or learning algorithms, which are necessary to understand the foundation model risk fully. BIML's 2024 paper states, "An LLM foundation model user is provided with what amounts to an undocumented, unstable API that sometimes exhibits unanticipated behavior" [6]. If you use a black box foundation model, analyze its security, operation, data, and output validation to verify it works as desired.

ii. Model Hub Risk

It is commonplace for developers to provide domain-specific models via popular model hubs, such as Hugging Face. Adversaries have targeted these model hubs to upload a model with malicious functionality to be used directly or indirectly via another benign model [30]. Even though safeguards such as malware scanning are implemented to ensure the safety of model uploads, care must be taken to analyze all third-party models before using them.

iii. Unsafe Plugins/Agents

LLMs' functionality is significantly expanded with plugins, allowing integrated applications to reason and solve complex prompts, obtain current information, and execute code. These plugins can be used as attack vectors by exploiting a lack of input/output validation, excessive privileges, and indirect prompt injection vulnerabilities. Such weaknesses can lead to data theft, remote code execution, sensitive information leakage, unauthorized data collection, and model poisoning.

If your application includes plugins, ensure these components are scanned continuously for vulnerabilities, implement input/output validation with authentication and access control, and patch regularly.

iv. Training Data Poisoning

In the context of LLMs, data poisoning is a manipulation of a dataset used in training to introduce weaknesses into an LLM. Data poisoning can happen unintentionally at any stage of the software development lifecycle by ingesting training datasets from unreliable public internet sources, fine-tuning, or user interactions with the LLM-integrated application once deployed. Data can be considered poisoned if it contains PII or sensitive information. In 2023, Stanford researchers showed that a popular dataset (LAION-5B) used to train image generation algorithms contained images related to child sexual abuse material [33].

LLM-based applications can generate output containing misleading information with false or toxic content, including biases, discrimination, and other harmful content. In these cases, the model provided an answer based on the information it has seen within the given context. If the model for its training recursively consumes this inaccurate content, it can poison the model/data by reinforcing the problematic data, increasing its magnitude. This feedback loop is referred to as "Recursive Pollution." This can occur unintentionally if a user's input prompt generates a problematic response that is then consumed by the model unbeknownst to the user. Adversaries can execute this attack to undermine the integrity of the LLM model, rendering the model and LLM-integrated application entirely useless. To mitigate data risks, track and protect datasets used for model development where possible with version control. External training data sources and model cards must be tracked with a machine learning bill of material (ML-BOM). LLMs should not use their output to avoid recursive pollution unless a data sanity check is done before ingestion. Incorporate humans in the feedback loop to avoid harmful content.

v. Model Trustworthiness

Datasets used for model development are too large for human review, may contain inaccurate, sensitive, or biased information, or could be poisoned or unavailable for review if present in a black box component, so the LLMs should be viewed as untrustworthy and scrutinized for use. Data are often encoded, filtered, and re-represented before use in model development using computer programs and filters driven by humans that can bias a model. LLMs' auto-associative predictive generation and stochastic nature, coupled with human overreliance, may increase the problems in model output if the output is further used in retraining the model. Model bias can occur during the training and deployment of LLM-integrated applications [31]. A few main ways it occurs are through selection bias, which is due to lack of representation of the entire population or target audience in training data; contextual bias due to the LLM failing to understand the context of a conversation or prompt; and linguistic bias from an LLM favoring specific languages, vocabularies, or cultural references over others.

Data used to train an LLM must be highly scrutinized at all stages for model development and in every interaction with an LLM in an LLM-integrated application to confirm it behaves as intended to prevent such biases.

I. Sensitive Information Disclosure

The dataset used for model development may include PII, business secrets, source code, domain-specific algorithms, other sensitive information, and so on by design or unintentionally. LLM applications can reveal this sensitive information via its output. Attackers can steal this data to clone the model for competitive advantage or reveal issues in the model such as sensitive data, ethical issues, and biases which make their way into the model due to lack of sanitization or knowledge and damage model reputation and/or LLM-integrated application. For example, Samsung employees used ChatGPT in three separate instances to check confidential source code for errors, requested code optimization, shared a meeting recording to convert into notes, and unintentionally leaked company sensitive information [29].

Protecting the LLM application's data is critical to maintaining model integrity and safeguarding sensitive information. Developers must leverage LLM-specific ML-BOM or model cards for dataset management, guardrails, or mitigation strategies to protect the data and prevent data leakage in output, as described in the above sections. In some cases, data removal may be needed by LLM model unlearning if sensitive data is not prevented from entering the model in the first place.

IV. APPLICATIONS

LLM-powered applications are utilized in many industries and are expected to grow. Some of the typical and emerging applications seen today are discussed below.

A. Customer Service

LLM applications have pervasively improved customer service with chatbots and automated email responses. Chatbots interact with customers in natural language, address customer queries, and provide information for further assistance. LLM-enabled customer support improves response time, enhances customer experience, and reduces workload on human support teams.

B. Education

Most recently, LLM applications have been used to personalize learning and provide tailored assistance to an individual student's learning style and pace. It can generate interactive reading content and adjust complexity to adapt to a student's comprehension level.

An application can act as a virtual tutor in answering students' questions, help with problem-solving steps, and encourage them with positive messages.

For example, Duolingo has features such as "Explain My Answer" and "Roleplay" to provide detailed explanations about students' responses and engage students to practice real-world conversation skills with virtual characters, providing personalized and interactive language learning.

C. Healthcare

The healthcare industry collects vast amounts of structured, unstructured, and semi-structured data daily.

This data comprises doctor's notes, electronic medical records, diagnoses, lab results, smart gadget metrics, medical imaging, etc. LLM-powered clinical decision support systems can analyze this data to extract insights on patient diagnoses and treatments, discover new medicines, and advance medical research.

D. Type of LLM-Based Applications

Common types of LLM-based applications are chatbots, copilots, and autonomous agents. Let us briefly look at each of them.

1. *Chatbots* primarily simulate interactive conversations with humans. They generate text to answer questions and support customers in customer service applications.

Examples:

- Domino's Pizza uses a chatbot to help customers order pizza.
- ChatGenius is a chatbot that enhances customer service with custom responses to user inquiries.

2. *Copilots* are LLM applications that mainly assist humans in performing tasks to become more productive. These include writing, coding, creating ideas, identifying errors, and improving their work.

Examples:

- Grammarly helps users with their writing, identifying grammar errors and suggesting improvements with feedback.
- GitHub Copilot helps programmers write code by generating code samples and assisting in debugging errors.

3. *Autonomous Agents* are LLM applications that go beyond assisting in tasks and functions to engage more through action using interconnected, automated systems with task decomposition and independent decision-making. This is an emerging trend and an area of ongoing research [35].

Examples:

- GPT-Engineer builds web application software and enhancements from natural language specification.
- ChemCrow is designed to accomplish tasks across organic synthesis, drug discovery, and materials design [32].

V. CONCLUSION

LLM-integrated applications will continue growing, providing significant value to users and businesses. In the future, LLM technology will have multi-modal capabilities, the ability to affect the physical world, perform reasoning, and interact with humans, expanding its application domains to healthcare, law, autonomous systems, and more. New attack vectors are constantly being discovered with this fast-evolving technology, making it challenging to secure LLM-integrated applications against all potential attacks. System designers and developers responsible for developing LLM-integrated applications will be accountable for choosing the LLM foundation model and building applications that are properly secured. Making your application robust to inherent natural and malicious errors can improve its security. By considering the risks discussed in this paper and applying the mitigations recommended, developers can reduce the exposure to these risks. The complete system requires following security best practices to help ensure a secure deployment.

DECLARATION STATEMENT

Funding	No, I did not receive it.
Conflicts of Interest	No conflicts of interest to the best of my knowledge.
Ethical Approval and Consent to Participate	No, the article does not require ethical approval and consent to participate with evidence.
Availability of Data and Material	Not relevant.
Authors Contributions	I am the sole author of the article.

REFERENCES

1. *Chatgpt*, openai.com/chatgpt/.
2. *CHATGPT Sets Record for Fastest-Growing User Base - Analyst Note | Reuters*, www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/.
3. "Command R." *Cohere*, cohere.com/models/command.
4. *Meta Llama*, ai.meta.com/llama/.
5. *GPT-4*, openai.com/research/gpt-4/.
6. McGraw, Gary, et al. *An Architectural Risk Analysis of Large Language Models: Applied Machine Learning Security*, berryvilleiml.com/docs/BIML-LLM24.pdf.
7. *Artificial Intelligence Definitions*, hai.stanford.edu/sites/default/files/2020-09/AI-Definitions-HAI.pdf.
8. "Generative AI Foundations: An Introduction to Basic Generative AI Concepts." *Sendbird*, sendbird.com/developer/tutorials/introduction-to-basic-generative-ai-concepts.
9. "Introduction to Deep Learning." *GeeksforGeeks*, 26 May 2024, www.geeksforgeeks.org/introduction-deep-learning/.
10. Team, Toloka. *Difference between AI, ML, LLM, and Generative AI*, 23 May 2024, toloka.ai/blog/difference-between-ai-ml-llm-and-generative-ai/.
11. "What Are Large Language Models?: Nvidia Glossary." *NVIDIA*, www.nvidia.com/en-us/glossary/large-language-models/.
12. Sajid, Haziqa. "A Comprehensive Overview of Large Language Models." *Wisecube*, 1 June 2023, www.wisecube.ai/blog/a-comprehensive-overview-of-large-language-models/.
13. *OWASP*, owasp.org/www-project-top-10-for-large-language-model-applications/assets/PDF/OWASP-Top-10-for-LLMs-2023-v1_1.pdf.
14. Smith, Craig S. "What Large Models Cost You – There Is No Free AI Lunch." *Forbes*, Forbes Magazine, 20 Feb. 2024, www.forbes.com/sites/craigsmith/2023/09/08/what-large-models-cost-you--there-is-no-free-ai-lunch/?sh=44cc5add4af7.
15. McGuinness, Patrick. "GPT-4 Details Revealed." *GPT-4 Details Revealed - by Patrick McGuinness*, AI Changes Everything, 12 July 2023, patmcguinness.substack.com/p/gpt-4-details-revealed.
16. "LLM Training: A Simple 3-Step Guide You Won't Find Anywhere Else!" *Medium*, Medium, 10 Mar. 2024, masteringllm.medium.com/llm-training-a-simple-3-step-guide-you-wont-find-anywhere-else-98ee218809e5.
17. "Fine-Tuning Large Language Models (Llms) in 2024." *SuperAnnotate*, www.superannotate.com/blog/llm-fine-tuning.
18. Ben Dickson, et al. "The Complete Guide to LLM Fine-Tuning." *TechTalks*, 15 Aug. 2023, bdtechtalks.com/2023/07/10/llm-fine-tuning/.
19. "Security Guardrails for LLM: Ensuring Ethical AI Deployments." *Security Guardrails for LLM: Ensuring Ethical AI Deployments*, www.turing.com/resources/implementing-security-guardrails-for-llms.
20. "LLMS Guardrails Guide: What, Why & How: Attri AI Blog: Attri.Ai Blog." *LLMs Guardrails Guide: What, Why & How | Attri AI Blog | Attri.Ai Blog*, attri.ai/blog/a-comprehensive-guide-everything-you-need-to-know-about-llms-guardrails.
21. Daryanani, Lavina. "How to Jailbreak Chatgpt." *Watcher Guru*, 7 Feb. 2023, watcher.guru/news/how-to-jailbreak-chatgpt.
22. Greshake, Kai, et al. "Not What You've Signed up for Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection." *arXiv.Org*, 5 May 2023, doi.org/10.48550/arXiv.2302.12173. https://doi.org/10.1145/3605764.3623985
23. Liu, Yi, et al. "Prompt Injection Attack against LLM-Integrated Applications." *arXiv.Org*, 2 Mar. 2024, doi.org/10.48550/arXiv.2306.05499.
24. Zou, Andy, et al. "Universal and Transferable Adversarial Attacks on Aligned Language Models." *arXiv.Org*, 20 Dec. 2023, doi.org/10.48550/arXiv.2307.15043.
25. "CHATGPT Plugin Exploit Explained: From Prompt Injection to Accessing Private Data · Embrace the Red." *Embrace The Red*, 28 May 2023, embracethered.com/blog/posts/2023/chatgpt-cross-plugin-request-forgery-and-prompt-injection./.
26. Xu, Ziwei, et al. "Hallucination Is Inevitable: An Innate Limitation of Large Language Models." *arXiv.Org*, 22 Jan. 2024, doi.org/10.48550/arXiv.2401.11817.
27. Garcia, Marisa. "What Air Canada Lost in 'Remarkable' Lying AI Chatbot Case." Forbes, Forbes Magazine, 19 Feb. 2024, www.forbes.com/sites/marisagarcia/2024/02/19/what-air-canada-lost-in-remarkable-lying-ai-chatbot-case/.
28. Russell, Josh. "Judge Won't Sanction Michael Cohen over Ai-Generated Fake Legal Cases." Courthouse News Service, 20 Mar. 2024, www.courthousenews.com/judge-wont-sanction-michael-cohen-over-ai-generated-fake-legal-cases/.
29. Mauran, Cecily. "Whoops, Samsung Workers Accidentally Leaked Trade Secrets via Chatgpt." *Mashable*, Mashable, 6 Apr. 2023, mashable.com/article/samsung-chatgpt-leak-details.
30. Huynh, Daniel. "POISONGPT: How to Poison LLM Supply Chainon Hugging Face." *Mithril Security Blog*, Mithril Security Blog, 18 Dec. 2023, blog.mithrilsecurity.io/poisongpt-how-we-hid-a-lobotomized-llm-on-hugging-face-to-spread-fake-news/.
31. Knapton, Ken. "Council Post: Navigating the Biases in LLM Generative AI: A Guide to Responsible Implementation." *Forbes*, Forbes Magazine, 5 Oct. 2023, www.forbes.com/sites/forbestechcouncil/2023/09/06/navigating-the-biases-in-llm-generative-ai-a-guide-to-responsible-implementation/.
32. Weng, Lilian. "LLM Powered Autonomous Agents." *Lil'Log (Alt + H)*, 23 June 2023, lilianweng.github.io/posts/2023-06-23-agent/.
33. Thiel, David. "Investigation Finds AI Image Generation Models Trained on Child Abuse." *FSI*, cyber.fsi.stanford.edu/news/investigation-finds-ai-image-generation-models-trained-child-abuse.

34. Crist, Carolyn. “Chatgpt Gives Incorrect Answers about Eoe.” Medscape, 16 Nov. 2023, www.medscape.com/viewarticle/998537?form=fpf.
35. Hague, Danny. “Multimodality, Tool Use, and Autonomous Agents: Large Language Models Explained, Part 3.” *Center for Security and Emerging Technology*, 25 Mar. 2024, cset.georgetown.edu/article/multimodality-tool-use-and-autonomous-agents/.
36. Ansari, M. Z., Ahmad, T., & Fatima, A. (2019). Feature Selection on Noisy Twitter Short Text Messages for Language Identification. In International Journal of Recent Technology and Engineering (IJRTE) (Vol. 8, Issue 4, pp. 10505–10510). https://doi.org/10.35940/ijrte.d4360.118419
37. Lalaei, R. A., & Mahmoudabadi, Dr. A. (2024). Promoting Project Outcomes: A Development Approach to Generative AI and LLM-Based Software Applications’ Deployment. In International Journal of Soft Computing and Engineering (Vol. 14, Issue 3, pp. 6–13). https://doi.org/10.35940/ijsce.d3636.14030724
38. Rao P, Mr. V., & Sivakumar, Dr. A. P. (2020). A Comprehensive Retrospection of Literature Reported Works of Community Question Answering Systems. In International Journal of Innovative Technology and Exploring Engineering (Vol. 9, Issue 3, pp. 1904–1907). https://doi.org/10.35940/ijitee.c8769.019320

AUTHOR PROFILE

Nikhil Pesati is a highly motivated student who is looking to further his knowledge in large language models and cybersecurity. He is a senior attending The Harker High School, a prestigious college preparatory school located in San Jose, California, where he has built a strong academic background in computer science and math through taking advanced courses like Compilers & Interpreters as well as Multivariate Calculus. Nikhil has been conducting research in large language models and cybersecurity for about three years. His prior research in the field includes experience with assessing the capabilities of LLMs as well as research on the discovery and classification of sensitive data. Aside from his academic accomplishments, Nikhil possesses a keen interest in soccer, both playing and watching. He also likes to play and develop computer games and has thus become the media director of the Game Development Club. Nikhil plans to pursue an undergraduate degree in computer science and engineering to make a meaningful impact on citizens’ welfare.