Feature: Grey Literature Survey on the Release Readiness Criteria of Generative AI-based Software Products

A State-of-the-practice Release-readiness Checklist for Generative AI-based Software Products

Harsh Patel, *Queen's University, Canada*

Dominique Boucher, PhD, *National Bank of Canada, Canada*

Emad Fallahzadeh, PhD, *Queen's University, Canada*

Ahmed E. Hassan, PhD, *Queen's University, Canada*

Bram Adams, PhD, *Queen's University, Canada*

Abstract—This paper investigates the complexities of integrating Large Language Models (LLMs) into software products, with a focus on the challenges encountered for determining their readiness for release. Our systematic review of grey literature identifies common challenges in deploying LLMs, ranging from pre-training and fine-tuning to user experience considerations. The study introduces a comprehensive checklist designed to guide practitioners in evaluating key release readiness aspects such as performance, monitoring, and deployment strategies, aiming to enhance the reliability and effectiveness of LLM-based applications in real-world settings.

Generative AI, especially Large Language Models (LLMs), are increasingly being integrated into software products [1], marking a significant shift in how companies approach product development and release. This integration is driven by tangible improvements in user productivity and creativity. Additionally, the substantial economic potential of generative AI is noteworthy, with an estimated contribution of $2.6 trillion to $4.4 trillion annually to the global economy, according to McKinsey's report[1].

However, determining the release-readiness of these products is increasingly complex. This involves ensuring compliance with user and safety requirements, passing quality assurance checks, evaluating model performance, ethical considerations, and the potential impact on users. Companies must also navigate evolving AI legislation across countries, further complicating the release process.

Many challenges of ensuring production-readiness of *generative* AI products are similar to *traditional* AI-based systems, including non-determinism and testing difficulties. These challenges highlight the need for release checklists—formal lists of evaluation criteria to determine if a product is ready for release to end users.

Several studies have explored release-readiness for traditional AI-based software, drawing on industry experiences like Google's testing rubric [2], which emphasizes reliability with a 28-test scoring system for assessing ML system production readiness. Zinkevich [3] presents a guide on ML system development best practices, spanning integration to feature engineering. Microsoft's production checklist [4] aids teams in evaluating ML model production readiness, focusing on performance, metrics, data quality, integration, and ethical considerations. Unlike these works, our study develops a release-readiness checklist specifically for generative AI-based software products.

Determining when generative AI-based software products are ready for release poses a more complex challenge. LLMs inherit typical ML concerns around data dependency and model unpredictability and face unique issues such as ensuring contextually accurate and unbiased language understanding, managing the evolving scope of human language, and addressing the

XXXX-XXX © 2024 IEEE
Digital Object Identifier 10.1109/XXX.0000.0000000

[1] https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier

ethical implications of their responses [12]. Companies have begun sharing their processes and experiences through blogs and industry conferences, contributing to a growing body of “grey literature” on the subject.

This paper synthesizes a release checklist for generative AI-based products. Unlike traditional AI checklists, our checklist is compiled from 65 grey literature sources across 44 organizations, identifying key release challenges. Our results will enable future automation of release-readiness evaluation steps.

METHODOLOGY

Our systematic grey literature review follows a similar approach for search construction as used by earlier work [13], and aims at capturing the breadth of discussions surrounding the deployment of LLMs in production environments. We started from a base query, i.e., “generative ai release checklist”, which covers the seminal resources in the field such as the “LLMs in Production Conference” [5] and the influential blog posts of Matt Bornstein [14] and Chip Huyen [6]. We then split the query into three parts (1. “generative ai”, 2. “release” and 3. “checklist”) and included synonyms or related words to improve the coverage of grey literature, yielding the resulting search query:

```
("generative ai" | "large language
models" | llms | "foundation models"
| chatgpt | "conversational ai") AND
(release | production | deployment |
monitoring | observability | operations)
AND (checklist | checks | guardrails
| considerations | requirements |
practices | patterns | challenges |
methods | risks)
```

Our search covered the period from June 1, 2018, the release date of GPT-1 by OpenAI, to September 1, 2023, the start of our survey.

Inclusion criteria of our search:

- Literature must be authored by or associated with organizations focused on the field of generative AI.
- Content should specifically discuss LLMs in production settings.
- Accessibility to the full text without restrictions.

The exclusion criteria:

- Product announcements, advertisements, and demos.
- News articles.
- Non-English literature.
- Duplicates.
- Literature that mentions LLM production without detailing the techniques used.
- Articles behind paywalls.

Our search yielded 522 million results, but relevance dropped after the top 100. We collected the top 100 results and augmented them with targeted searches for the top 10 companies in generative AI and LLMs, as recognized by sources like State of AI [10] and Gartner [11]. This approach netted 1,100 results, narrowed down to 35 through our criteria, plus 30 more from snowball sampling.

The first and third authors developed a taxonomy for release-readiness challenges and solutions using a grey literature sample. They independently coded 20% of the sample, then merged their taxonomies to form a consensus. This taxonomy was reapplied to re-code the initial sample, with comparisons for consistency and conflict resolution. Krippendorff’s Kappa score of 0.917 indicated high inter-rater reliability (> 0.8) [15]. This agreement allowed the first author to code the remaining 80% of the sources. The findings were organized into a mindmap of release-readiness challenges and solutions, complemented by references. Workshops were conducted with the team to refine the mindmap into a checklist, incorporating team feedback.

Our replication package[2] includes the resulting mind map to visually organize the findings, the collected notes for each grey resource we reviewed and an online repository of references for further exploration. We refer to online references in this paper with the notation “O«number»”.

RESULTS: CHALLENGES

Figure 1 lists the 31 identified release challenges related to generative AI, and their prevalence in the grey sources. Note that the latter measures the number of “mentions” of these challenges in the grey sources, which is not necessarily equal to the actual frequency of challenges in practice (companies may use specific strategies or face challenges without necessarily discussing them in public media). The top five challenges discussed are:

1) **Reliability:** Ensuring that LLMs consistently deliver accurate and coherent responses, avoiding hallucinations, which are instances where the model generates incorrect or nonsensical information.
2) **Deployment Resource Management:** Optimizing computational resources for LLMs in production to improve cost, latency, and performance.

[2]https://github.com/SAILResearch/replication-24-harsh-generative-ai-release-readiness-checklist

3) **Managing Embeddings:** Handling the creation, storage, and update of text documents' vector representations to improve semantic understanding and response relevance.
4) **Pre-deployment Evaluation:** Conducting comprehensive testing to ensure LLMs' consistency, unbiasedness, and safety across demographics before release.
5) **Prompt-centric Orchestration:** Effectively designing and managing prompts to utilize LLMs' full potential, addressing task decomposition, external tool integration, and sensitive information protection for privacy and security.

RESULTS: RELEASE-READINESS CHECKLIST

This section presents a release-readiness checklist for LLM-based software, derived from our grey literature findings. It follows the software lifecycle stages shown in Figure 1's flowchart, highlighting key challenges at each stage. The checklist provides checkpoints for assessing an LLM-based product's readiness and identifies improvement areas, including references and effective practices for each challenge. Checklist items are labeled with "C" followed by a number (e.g., C1, C2, C3), with subpoints (e.g., C1.1, C1.2) detailing specific actions. Practitioners should implement these practices or suitable adaptations for each relevant challenge. Note that these grey literature suggestions require further evaluation and verification, as they might not be concrete solutions.

C1. When pretraining a new LLM:

C1.1 Check if your LLM needs to be trained using sensitive/private data provided by others. [O43]
This consideration is crucial for ensuring privacy compliance and the ethical use of data. If direct data sharing is required, it is recommended to use Federated Learning (FL). FL shares models with clients to train on their data locally, ensuring the data never leaves the source device. The updated model parameters are then aggregated from all clients, incorporating the collective insights while maintaining data privacy.

C1.2 Partition the training process across multiple accelerators (e.g., GPUs or other devices) to speed up training. [O28][O42]
Distributed platforms for LLM workloads, such as Ray, are essential for enabling parallelization and acceleration, especially when incorporating advanced techniques like FL.

C1.3 Anonymize data. [O43]
To ensure data privacy while still using it for training purposes, it is recommended to anonymize the data before it is sent or used elsewhere. Use tools like Gretel.ai, Private AI, and Tonic.ai to alter or remove identifying information. Complement these automated processes with manual checks to catch any nuances that automation might miss, thus preventing the association of data with individuals.

C1.4 Mitigate training data poisoning risks. [O26][O27][O29]
Attackers may deliberately introduce misleading information into training datasets, causing LLMs to learn from biased or incorrect data. Verify the sources of external training data used by your organization and maintain detailed records of its origins, similar to the methods used in the Software Bill of Materials (SBOM). For more information on emerging AI and data SBOM scenarios, please refer to the SPDX specs[3].

C2. When fine-tuning an LLM:

In addition to the checklist items related to pre-training, fine-tuning also requires the following items.

C2.1 Minimize unsafe behaviours or align the loss function to complex human values. [O2][O25][O29][O58]
This is achieved by integrating instruction fine-tuning with reinforcement learning through Reinforcement Learning with Human Feedback (RLHF). This approach uses human preferences, such as pairwise comparisons, to train a reward model, ensuring AI actions are ethical, appropriate, and aligned with human expectations, thereby enhancing the safety and reliability of AI applications.

C2.2 Utilize efficient fine-tuning processes to improve training efficiency. [O2][O6][O9]
As models grow in size, full fine-tuning on prosumer hardware becomes impractical, and storing separate fine-tuned models for each task is costly due to their large size. Parameter Efficient Fine-Tuning (PEFT) methods address these challenges by fine-tuning only a few additional parameters while freezing most of the pre-trained LLM's parameters. This approach significantly reduces computational and storage costs and mitigates issues like catastrophic forgetting observed in full fine-tuning.

[3]https://fossa.com/blog/spdx-3-0/,2023

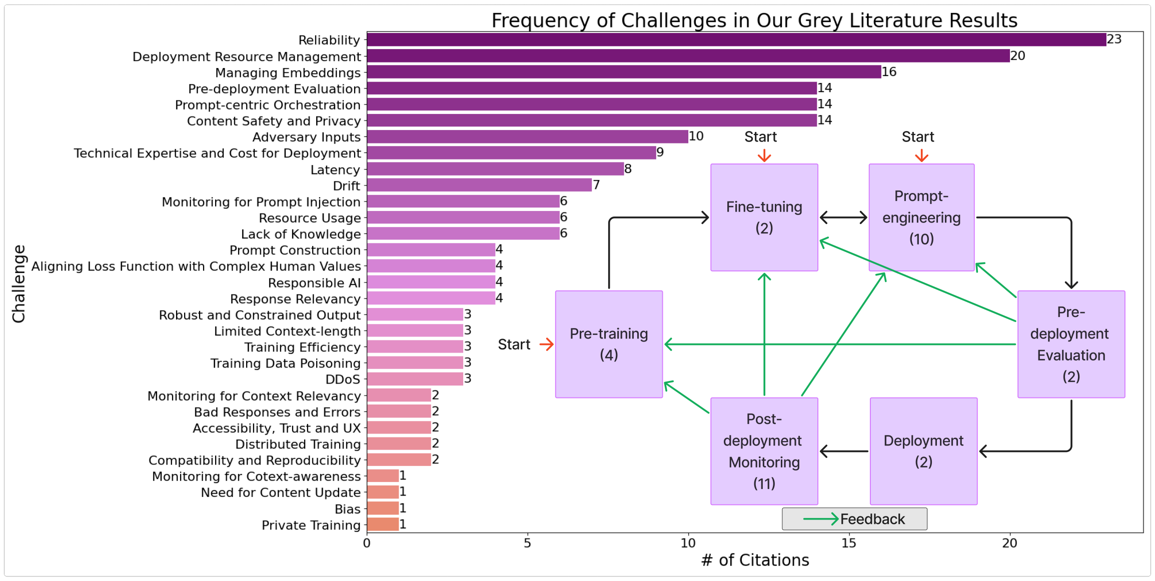

FIGURE 1. Frequency of challenges in our grey literature results (The numbers in the flowchart indicate the number of challenges associated with each stage of the LLM lifecycle)

C3. When doing prompt engineering:

C3.1 Consider different prompt construction methods. [O1][O4][O7][O10][O24][O47]
Exploring methods like Zero-shot, where the model tackles a task with no prior examples, Few-shot, where a few examples are provided for context, Chain-of-Thought (CoT) prompting, which encourages step-by-step reasoning, Self-Consistency, which selects the most accurate response from multiple outputs, and Tree-of-Thought[4], which structures reasoning as a tree with multiple branches, can significantly enhance LLM effectiveness. Crafting precise prompts with clear instructions or questions and integrating contextual inputs or examples also improves response quality and relevance.

C3.2 Utilize known practices to improve reliability and hallucinations. [O1][O2][O3][O4][O7][O9][O13][O14][O18][O19][O22][O24][O26][O34][O36][O38][O40][O44][O46][O47][O57][O60][O63]
Ensuring consistent outputs for identical inputs is crucial for user trust in LLMs, which can exhibit non-deterministic behavior. Solutions include model-based evaluation or self-evaluation techniques, such as OpenAI Evals, where an LLM assesses its own outputs for consistency, reliability, and ethical appropriateness. Another method is self-consistency and multiple prompting, generating multiple responses for the same input and determining the final output through majority voting or LLM selection using APIs like OpenAI's. Guardrails are also essential to ensure outputs are coherent, accurate, factual, and free from harmful content, safeguarding against adversarial inputs; examples include Guardrails.ai, NeMo, Guidance, and rellm.

C3.3 Bridge knowledge gaps by enriching prompts with relevant context. [O38][O47][O55][O56][O63]
LLMs can sometimes make incorrect assumptions to fill knowledge gaps, leading to hallucinations. These knowledge gaps often arise from insufficient context or ambiguous prompts. Utilizing Retrieval-Augmented Generation (RAG) techniques addresses this issue by providing richer context for prompts. By grounding model responses in factual content and supplying additional context to clarify ambiguous terms or references, RAG significantly reduces the occurrence of hallucinations and improves response accuracy.

[4]This technique is not present in our mindmap.

C3.4 Utilize vector stores[5] and embedding models[6] to store, search, and update vector representations[7] of language data. [O2][O5][O7][O9][O10][O11][O14][O17][O33][O36][O38][O44][O55][O60][O61][O63]

Vector stores and embedding models are essential for managing and querying vector representations of language data, enabling efficient retrieval of relevant information (see Section C3.3) and reducing hallucinations. Embeddings represent critical implementation decisions impacting RAG techniques, which rely on effective data management to provide enriched context for prompts. Platforms like Weaviate, Pinecone, and pgvector for vector stores, along with embedding models from OpenAI, Huggingface, and Cohere, play a crucial role in creating, storing, and updating vector representations of language data, ensuring more accurate and contextually relevant responses.

C3.5 Employ efficient context-retrieval techniques to overcome LLM context length limit. [O1][O4][O38]

Large models like GPT-4 see performance decline near their context window limit, affecting both inference time and accuracy. Implementing chunking strategies to deliver the most relevant documents or segments enhances retrieval efficiency. Additionally, applying contextual compression to condense and summarize key facts increases the density of useful information within the LLM's context window. However, these methods can sometimes produce invalid or incomplete results, so verification is necessary.

C3.6 If your application is used by untrusted users, it is crucial to address adversarial inputs. [O2][O4][O18][O19][O34][O26][O29][O40][O54][O57]

LLMs must be safeguarded against prompts intended to manipulate model output. Implementing clear instructions within prompts is a fundamental defensive strategy. Utilizing Guardrails (see Section C3.2) and adapting SQL injection defense strategies by parameterizing prompt components to separate instructions from inputs further enhances security.

C3.7 Ensure prompts' compatibility across changes to the underlying LLM (e.g., migrating from GPT-3.5 to GPT-4) and reproducibility across changes to the prompt itself. [O1][O19]

In prompt engineering, prompts for older models often need revisions for new versions, unlike traditional software updates. Therefore, unit-testing all prompts with evaluation examples before model changes is crucial. Temperature settings ensure consistent outputs but reduce creativity. Prompt versioning is essential for tracking the impact of changes, requiring versioning and performance monitoring. Using prompt templates offers a scalable way to craft prompts, incorporating instructions, few-shot examples, or action sequences tailored for language models.

C3.8 Incorporate practices to ensure content safety and privacy. [O1][O2][O4][O5][O9][O18][O24][O26][O27][O38][O40][O57][O60]

Review and adjust prompts to mitigate risks in third-party LLM-generated content, especially after RLHF fine-tuning (see Section C2.1). This helps prevent harmful or sensitive information and ensures compliance with ethical and privacy standards. Implement profanity detection models like the PyPI module "profanity-check" to filter inappropriate language. Use LLMs to evaluate responses for inappropriate content, and integrate PII masking solutions like LangChain or LLamaIndex, along with PII detection tools such as Microsoft/presidio or Azure services, to enhance content safety and privacy measures.

C3.9 Consider reusing prompts, decomposing large tasks into smaller ones, utilizing external tools, or chaining prompts. [O7][O10][O11][O17][O36][O38][O40][O44][O55][O61][O62]

Prompt orchestration enhances LLM effectiveness by reusing prompts, simplifying problems, integrating external tools, and sequencing prompts for multi-step tasks. Frameworks like LangChain and LlamaIndex excel in managing prompt chaining, API interactions, contextual data retrieval, and memory across LLM calls. They also use thought decomposition methods such as CoT, Tree of Thought, or ReAct to break down complex tasks into manageable sub-tasks, improving systematic reasoning and response generation.

C3.10 Ensure responses are highly constrained using types, templates, and constraints.

[5]Vector stores are specialized databases optimized for storing and querying high-dimensional vectors, allowing for efficient similarity searches and retrieval.

[6]Embedding models are machine learning models that convert data, such as words or sentences, into dense vector representations that capture semantic meaning.

[7]Vector representation refers to the numerical encoding of data into a multi-dimensional space, where similar items are positioned close to each other.

[O61][O62][O63]
In systems combining LLMs with other components, maintaining response clarity is essential. Constrained decoding mitigates prompt injection risks by enforcing token-level constraints, thus enhancing security (LMQL). Unlike guardrails, it controls each output step for validity. For more details and examples, visit the prompting guide website[8].

C4. Pre-deployment evaluation ensuring that a model's performance matches safety and security criteria for application deployment.

C4.1 Evaluate your application's performance pre-deployment. [O2][O5][O6][O9][O12][O15][O17][O19][O35][O36][O38][O55][O56][O58]
Measure model performance against benchmarks to ensure standards. Use A/B testing to compare responses and prompt designs in real-world scenarios. For classification tasks and extractive QA, apply metrics like recall, precision, and PRAUC. For tasks without clear answers, such as translation or summarization, use BLEU, ROUGE, or semantic similarity measures like BERTScore. Conduct human evaluations to capture nuances missed by automated metrics. Perform Penetration Tests and Red Teaming to identify biases or issues, leveraging resources like Anthropic's adversarial simulations[9]. Develop task-specific benchmarks with Eval Driven Development (EDD) for tasks like summarization or dialogue, focusing on metrics that reflect each task's unique aspects to ensure relevance and effectiveness. Use frameworks like G-Eval for LLM self-evaluation to enhance transparency and interpretability in assessments.

C4.2 Consider implementing practices to ensure accessibility and foster end-user trust in your application. [O2][O42]
For comprehensive guidance on creating human-centric AI interactions, consult AI UI/UX guidelines from industry leaders like Microsoft[10], Google[11], and Apple[12].

[8]https://www.promptingguide.ai
[9]https://github.com/anthropics/hh-rlhf
[10]https://www.microsoft.com/en-us/research/publication/guidelines-for-human-ai-interaction/
[11]https://pair.withgoogle.com/guidebook/
[12]https://developer.apple.com/design/human-interface-guidelines/machine-learning

C5. LLM-related Deployment Considerations

C5.1 Consider whether a commercial or open-source LLM deployment suits your needs. [O9][O19][O26][O27][O29][O33][O36][O37][O41]
This decision parallels the choice between commercial and open-source software, weighing factors such as ease of deployment, support options, cost considerations, and flexibility for innovation. Commercial LLMs streamline deployment and provide robust support, focusing on application development despite higher costs and potential trade-offs in flexibility. In contrast, open-source LLMs require more technical expertise and infrastructure investment but offer extensive customization opportunities and cost efficiency, backed by a vibrant community for ongoing development and issue resolution. While prototyping and pre-deployment activities might involve experimentation with more expensive, state-of-the-art models like the latest GPT versions, the final deployment decision should balance cost and performance for sustainable operation.

C5.2 Optimize your application for latency, cost, and resource efficiency. [O2][O3][O4][O5][O9][O17][O19][O20][O24][O27][O28][O32][O33][O36][O38][O41][O47][O60][O63][O64]
When choosing third-party LLMs, balance the evolving costs and impacts of token size with the advantages of cloud scalability versus the security benefits of on-premise deployment. Implement semantic caching to minimize latency and computational overhead by reusing responses through efficient similarity algorithms. Employ techniques such as model compression, quantization, pruning, and distillation to enhance memory efficiency and computational speed. Model compression reduces the model size without significantly sacrificing performance, quantization converts parameters from high-precision to lower precision to save memory and computation, pruning eliminates less critical model parts, and distillation transfers knowledge from a larger model to a smaller one while maintaining performance. Additionally, it is worth considering the trade-off between using a longer prompt with a larger model versus fine-tuning a smaller model to achieve similar performance, as this can impact both cost and latency. Ensure hardware configurations (GPUs, TPUs, CPUs) match processing, memory, and storage needs. Explore smaller, specialized LLMs for specific

tasks and deploy memory optimization and distributed inference methods, including data and tensor parallelism. Techniques like FlashAttention and PagedAttention can help reduce memory usage and improve responsiveness. Lastly, optimize request scheduling to manage variable latency and ensure seamless user interactions with LLM-powered applications.

C6. Post-deployment monitoring to set up feedback loops and improve application performance.

C6.1 Monitor for prompt injection attacks. [O7][O16][O18][O26][O38][O40]
Detecting adversarial prompt inputs post-deployment is crucial to prevent manipulation of the LLM's outputs and avoid unintended interactions. Utilize an adversarial prompt detector like "rebuff" to identify and filter such inputs, leveraging LLMs' capabilities in specialized tasks like knowledge generation [8] and self-verification [9]. Additionally, analyze text similarity between known attacking prompts and current inputs to detect and mitigate potential threats effectively.

C6.2 Monitor the application for model resource usage. [O3][O9][O26][O32][O35][O63]
Continuously assess model performance, resource consumption, and cost efficiency to fine-tune operations post-deployment. Keep track of LLM token count and utilization to ensure cost-effective operation. Regularly monitor system resources such as CPU, GPU, and memory usage to uphold performance standards and make adjustments as needed. Potential LLM performance assessing metrics like total-tokens-per-second and time-to-first-token were not mentioned in the grey resources.

C6.3 If your application is publicly exposed, consider implementing solutions to prevent DDoS attacks. [O3][O8][O25]
Implement API rate limiting and use Captcha mechanisms to safeguard user experience and deter misuse by regulating access effectively.

C6.4 Monitor the application for model drift. [O1][O10][O16][O35][O38][O42][O43]
This involves addressing performance decline due to shifts in data distribution or user interaction patterns by monitoring input and output data. Use historical performance data as a benchmark to identify and address drift in model behavior or data distribution. Evaluate discrepancies between expected and actual prompts to adapt and refine LLM interactions based on real-world use. Implement concept drift detection strategies and cluster analysis using embeddings (see C3.4) to identify and correct drift issues post-deployment.

C6.5 Ensure your context-retrieval system serves the most "relevant" documents. [O14][O38]
This requires ensuring that the context-retrieval system accurately matches user queries with relevant documents, especially for unique or specific requests. Measure query density to evaluate if the vector store accurately represents user queries and adjust as needed to improve data relevance. Significant drift in query density indicates that the vector store lacks closely related data. Utilize ranking metrics to assess and enhance the precision of the search and retrieval process, ensuring users receive the most pertinent information.

C6.6 Evaluate the context provided to the LLM by the application. [O58]
Context evaluation is crucial in LLM applications to ensure response credibility, focusing on how the LLM uses the prompt, guiding information, and its knowledge base to generate accurate and relevant outputs. Implement secondary LLMs for cross-evaluation of context relevance, quantifying response integrity. Critically assess how LLMs use context to ensure factual accuracy and appropriateness, verifying the LLM's understanding of the topic and the accuracy and relevance of referenced or provided information.

C6.7 Continuously assess the necessity of content updates for the context-retrieval system. [O38]
If the LLM fails to answer certain queries, it may indicate a need to update the vector store (see C3.4). To address this, track the frequency of LLM failures to respond to prompts to establish a rejection metric.

C6.8 Verify the relevance of LLM responses. [O16][O35][O56][O58]
This entails regularly checking whether LLM outputs align with expected topics and maintain appropriate sentiment in relation to the end user. Track changes in LLM response sentiment to ensure consistency with expected topics, tone, and relevance for appropriate interactions. Ensure responses remain pertinent to predefined topics (e.g., politics). Analyze semantic similarity between queries and responses to confirm the LLM accurately addresses user intents.

C6.9 Ensure fairness. [O35][O40][O42][O43]
LLMs can inherit and propagate biases from their training data, so organizations need to track and

measure these biases using fairness metrics that vary by domain, such as gender, race, or other unintentional biases. One solution is to use sentiment scores (see C6.8). Additionally, assess the model's performance across various demographic groups; for example, measure gender bias by comparing performance across different genders. Combine toxicity classifiers with sentiment analysis (see C6.8) to identify and mitigate harmful content in LLM outputs, ensuring they are unbiased, safe, and respectful.

C6.10 Monitor the application for model latency. [O10][O11][O32][O35][O36][O42][O57][O63]
Maintaining latency within acceptable limits is essential for ensuring a positive user experience and operational efficiency in LLMOps. Utilize observability tools to detect high-latency prompts by analyzing API latency metadata, enabling targeted optimizations to improve response times.

C6.11 Monitor the application for bad model responses. [O35][O40]
Occasionally, LLMs may generate unwanted outputs. It is crucial to consider both implicit and explicit user feedback as pivotal indicators for monitoring, particularly negative or confused reactions.

THREATS

External Validity

Our checklist may not cover all nuances of LLM readiness due to the rapid advancements in the field since our survey. Despite this, the core challenges and solutions remain relevant. Consider the checklist a foundational tool for practitioners to build upon and adapt to the changing LLM technology landscape.

Internal Validity

Our checklist was created by a team of human coders, which may introduce bias. We attempted to mitigate this by using an established empirical methodology.

CONCLUSION

This paper investigates the release readiness of software products integrated with LLMs, synthesizing a comprehensive checklist to guide practitioners in evaluating their LLM products' readiness for release. As the generative AI landscape rapidly evolves, this checklist underscores the need for ongoing adaptation and community engagement to ensure the responsible and effective use of LLMs in software development.

ACKNOWLEDGMENTS

We are grateful to Mitacs and the National Bank of Canada for their support for our research.

REFERENCES

1. Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., Luo, X., Lo, D., Grundy, J. & Wang, H. Large language models for software engineering: A systematic literature review. *ArXiv Preprint ArXiv:2308.10620*. (2023)
2. Breck, E., Cai, S., Nielsen, E., Salib, M. & Sculley, D. The ML test score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference On Big Data (Big Data)*. pp. 1123-1132 (2017)
3. Zinkevich, M. Rules of machine learning: Best practices for ML engineering. *URL: Https://developers. Google. Com/machine-learning/guides/rules-of-ml*. (2017)
4. ISE, M. ML model production checklist - Engineering Fundamentals Playbook — microsoft.github.io. (https://microsoft.github.io/code-with-engineering-playbook/machine-learning/ml-model-checklist/,0), [Accessed 21-02-2024]
5. Community, M. LLMs in Production Conference - Event | MLOps Community — home.mlops.community. (https://home.mlops.community/public/events/llms-in-production-conference-2023-04-13,0), [Accessed 21-02-2024]
6. Huyen, C. Building LLM applications for production — huyenchip.com. (https://huyenchip.com/2023/04/11/llm-engineering.html,0), [Accessed 21-02-2024]
7. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A. & Zhou, D. Self-consistency improves chain of thought reasoning in language models. *ArXiv Preprint ArXiv:2203.11171*. (2022)
8. Liu, J., Liu, A., Lu, X., Welleck, S., West, P., Bras, R., Choi, Y. & Hajishirzi, H. Generated knowledge prompting for commonsense reasoning. *ArXiv Preprint ArXiv:2110.08387*. (2021)
9. Weng, Y., Zhu, M., He, S., Liu, K. & Zhao, J. Large language models are reasoners with self-verification. *ArXiv Preprint ArXiv:2212.09561*. (2022)
10. Benaich, N. & Capital, A. State of AI Report 2023 — stateof.ai. (https://www.stateof.ai/,0), [Accessed 28-02-2024]
11. Gartner Generative AI: What Is It, Tools, Models, Applications and Use Cases. (https://www.gartner.com/en/topics/generative-ai,0), [Accessed 28-02-2024]

12. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S. & Others Gpt-4 technical report. *ArXiv Preprint ArXiv:2303.08774*. (2023)
13. Raulamo-Jurvanen, P., Mäntylä, M. & Garousi, V. Choosing the right test automation tool: a grey literature review of practitioner sources. *Proceedings Of The 21st International Conference On Evaluation And Assessment In Software Engineering*. pp. 21-30 (2017)
14. Bornstein, M. & Radovanovic, R. Emerging Architectures for LLM Applications. (https://a16z.com/emerging-architectures-for-llm-applications/,2023), [Accessed 01-03-2024]
15. Krippendorff, K. Testing the reliability of content analysis data. *The Content Analysis Reader*. pp. 350-357 (2009)

Harsh Patel is currently pursuing a Master's degree at Queen's University, focusing on Software Engineering for Artificial Intelligence (SE4AI). His research primarily explores the release readiness of ML-based software products, with a special emphasis on post-deployment model recycling strategies. These strategies are designed to enable the efficient reuse of outdated models while maintaining their readiness for release. Harsh brings three years of software engineering experience to his research, having proficiency in technologies like Splunk, Python, JavaScript, Docker, and ReactJS. He is committed to advancing the integration of AI in software development processes. Contact him at patel.h@queensu.ca

Dominique Boucher, PhD is currently Senior Director, Conversational AI Technologies at National Bank of Canada where he is responsible for the deployment of NBC dialogue systems. His main interests revolve around the use of conversational interfaces to help optimize business processes. He has been in the speech recognition and conversational AI industry for more than 25 years and holds a PhD from the University of Montreal.

Emad Fallahzadeh, PhD is a Postdoctoral Researcher at Queen's University in the Software Analysis and Intelligence Lab, specializing in mining software repositories. His research interests involve applying machine learning techniques and large language models to address various software engineering challenges. He is a member of the Association for Computing Machinery (ACM) and has contributed to numerous research endeavors in the field of software engineering, including publications in the International Conference on Software Engineering (ICSE) and the Foundations of Software Engineering (FSE) conference proceedings. Contact him at emad.fallahzadeh@queensu.ca

Ahmed E. Hassan, PhD is the NSERC/RIM Industrial Research Chair in Software Engineering for Ultra Large Scale systems at Queen's University, Canada. He spearheaded the organization and creation of the Mining Software Repositories (MSR) Conference and its research community. He co-edited special issues of the IEEE Transactions on Software Engineering and the Journal of Empirical Software Engineering on the MSR topic. Early tools and techniques developed by his team are already integrated into products used by millions of users worldwide. His industrial experience includes helping architect the Blackberry wireless platform at RIM, and working for IBM Research at the Almaden Research Lab and the Computer Research Lab at Nortel Networks. He is the named inventor of patents in several jurisdictions around the world, including the United States, Europe, India, Canada, and Japan. He is a member of the IEEE.

Bram Adams, PhD is a full professor at Queen's University. His research interests include software release engineering (pre- and post-AI) and mining software repositories. His work has received the 2021 Mining Software Repositories Foundational Contribution Award. In addition to co-organizing the RELENG International Workshop on Release Engineering from 2013 to 2015 (and the 1st/2nd IEEE Software Special

Issue on Release Engineering), he co-organized the first editions of the SEMLA event on Software Engineering for Machine Learning Applications. He has been PC co-chair of SCAM 2013, SANER 2015, ICSME 2016 and MSR 2019, and ICSE 2023 software analytics area co-chair. He is a Senior IEEE Member. Contact him at bram.adams@queensu.ca