Meta

Llama

Responsible Use Guide

Resources and best practices for responsible development of products built with large language models

Contents

We’re excited to develop and release the latest Llama 3.2 models, the next generation of Llama.

Llama 3.2 includes a collection of multilingual text based pretrained and instruction tuned generative models in 1B and 3B parameter sizes for easy integration into devices, and 11B and 90B image reasoning models, our first multimodal models. For more information, please refer to the Llama 3.2 Model Card.

Instruction tuned text only models are intended for assistant-like chat and agentic applications like knowledge retrieval and summarization, mobile AI powered writing assistants and query and prompt rewriting. Building with new capabilities requires specific considerations in addition to the best practices that generally apply across all Generative AI use cases. Given the smaller size of our 1B and 3B models, not all use cases would be appropriate for these models. Developers should refer to the Llama 3.2 Model Card for the intended use cases. The responsibility considerations section in the guide walks you through recommendations on how to safely engage with applicable capabilities and apply the necessary system-level safeguards.

How to use this guide

This guide is a resource for developers that outlines common approaches to building responsibly at each level of an LLM-powered product. It covers best practices and considerations that developers should evaluate in the context of their specific use case and market. It also highlights some mitigation strategies and resources available to developers to address risks through alignment strategies on the model and system level. These best practices should be considered holistically because strategies adopted at one level can impact the entire system.

The recommendations included in this guide reflect current research on responsible generative AI. We expect these to evolve as the field advances and access to foundation models grows, inviting further innovation on AI safety. Decisions to implement best practices should be evaluated based on the jurisdiction where your products will be deployed and should follow your company's internal legal and risk management processes.

Overview of responsible AI and system design

Responsible AI considerations

Helping to ensure that generative AI technology does not produce content that could cause harm is of paramount importance. Generative AI is developing rapidly and is being driven by research, open collaboration, and product releases that are putting this technology in the hands of people globally. Growth at this scale presents novel challenges for the responsible deployment of AI, yet many of the principles of responsibility remain the same as for any other AI technology. These considerations, core to Meta's approach to responsible AI, include fairness and inclusion, robustness and safety, privacy and security, and transparency and control, as well as mechanisms for governance and accountability. LLMs are one of many AI tools, and their risks should be evaluated through these lenses according to how they will be used.

Foundation models and generative AI systems represent advancements in power and accuracy compared to predecessor technologies. The increase in the performance, utility, and flexibility of these models will likely lead to their ubiquity, as the value they bring to some pre-existing use cases may outweigh operational costs of deploying the systems. The ability to generate completely new content also opens up new use cases that must be evaluated for the types of risks they may present. There are potential risks related to the misuse of this technology that have already surfaced online, such as the creation or proliferation of illegal content, content which may be objectionable or hateful, or content that may result in the provision of unqualified advice. These instances may increase as generative AI tools become more accessible.

For our own, on-platform generative AI offerings, Meta is implementing safety measures to address context-specific risks. These mitigations are layered across different intervention points beyond those that can be assessed and mitigated in the foundation model.

Some mitigations applied at early stages in the development process can be detrimental to the performance and safety of the model, and some risks may be better addressed at later points in the product development cycle. Our vision for layered safety helps to empower developers to make decisions about balancing these trade-offs. Developers of generative AI-powered features that leverage open source models will have more power to ensure that their products are safe and benefit end users, while taking a holistic view of responsible AI across the entire product development cycle.

Mitigation points for LLM-powered products

A foundation model is a general purpose AI technology whereas an LLM-powered product has a defined use case and performs specific tasks to enable an intended use or capability through a user interface, sometimes embedded in products. An LLM-powered system encompasses both the foundation model and accompanying input-output safeguards, and a number of product-specific layers. At various points in the product development lifecycle, developers make decisions that shape the objectives and functionality of the feature, which can introduce potential risks. These decision points also provide opportunities to mitigate potential risks. It is critical that developers examine each layer of the product to determine which potential risks may arise based on the product objectives and design, and implement mitigation strategies accordingly.

Model-level safety: Model-level safety concerns the data preparation and processing best practices and human feedback or alignment practices for safety at the foundation and fine-tuned model level.

System-level safety: System-level safety is the venue for the most context-specific safety mitigations dependent on user interactions. Developers looking to craft safety mitigations specifically for their use case with the goal of offering their users the best product experience should leverage these options.

You can learn more about our layered approach to safety by visiting our resources for Llama trust and safety.

The following section presents responsible AI considerations for the different stages of LLM product development. At each of these levels, we highlight best practices for mitigating potential risks.

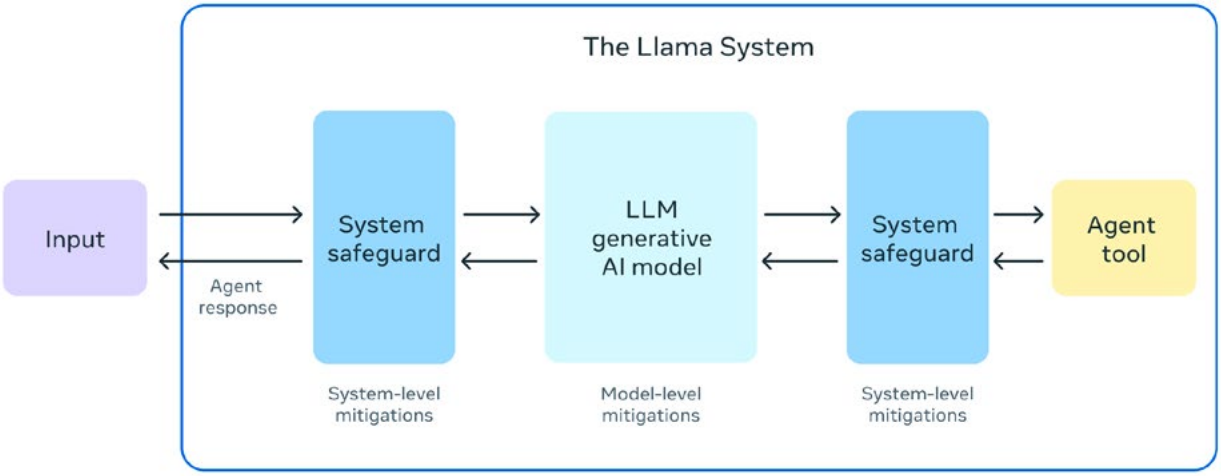


Figure 1: Overview of LLM model with system and model level mitigations

Development of the foundation model

During pre-training, a model builds its understanding of the statistical patterns across the sample of human language contained in its training data. After pre-training, the model can reproduce everything from simple grammatical rules to complex nuances like context, sentiment, and figurative language.

However, the model does not gain knowledge or generate beliefs about the world in the way humans do. It only learns to predict the next word in a sentence based on the patterns in its training data.

If you're going to use the pre-trained model, we recommend tuning it by using the techniques described in the next section to reduce the likelihood that the model will generate unsafe outputs that are in conflict with your intended use case and tasks. If you have terms of service or other relevant policies that apply to how individuals may interact with your LLM, you may wish to fine-tune your model to be aligned with those policies. It may also be necessary to establish new terms of service and policies specific to LLMs, or notify users about how their data or feedback provided will be used in fine-tuning.

Responsible LLM product development stages

Developers will identify a specific product use case for the released model, and are responsible for assessing risks associated with that use case and applying best practices to ensure safety. This section outlines the considerations and mitigation strategies available at each stage of product development and deployment.

At a high level these stages include:

1. Determine use case
2. Model-level alignment
3. System-level alignment
4. Build transparency and reporting mechanisms in user interactions

Determine use case

An important decision in the development process is which use case(s) to focus on. Most developers using this guide already have a use case in mind, such as customer support, AI assistants, internal productivity tools, entertaining end-user experiences, or research applications. If you're a developer who is not certain of a particular use case for which you would want to use the model, consider focusing on use cases that improve the lives of people and society, taking into consideration different ethical principles and values. Developing or adopting an internal risk assessment process can help identify potential risks for a specific use case and should focus on how your product's end users and others could be affected. This understanding is critical for evaluating in-context safety for your product deployment, and can take forms such as surveys and interviews of potential users or market analysis of similar product applications.

If you are new to considerations of values in the development and deployment of AI, refer to the principles and guidance on risk management released by academic and expert institutions, such as:

- OECD's AI Principles
- NIST's Trustworthy and Responsible AI Resource Center

Define content policies

Based on the intended use and audience for your product, a content policy will define what content is allowable and may outline safety limitations on producing illegal, violent, or harmful content. These limits should be evaluated in light of the product domain, as specific sectors and regions may have different laws or standards. Additionally, the needs of specific user communities should be considered as you design content policies, such as the development of age-appropriate product experiences. Having these policies in place will dictate the data needed, annotation requirements, and goals for safety fine-tuning, including the types of mitigation steps that will be implemented. Defining these policies will be used for labeling data in later stages when using RLHF and in additional product layers, such as making enforcement decisions for user inputs and model outputs.

If you are new to considerations of content policies, refer to commonly used policies in the industry such as the the taxonomy proposed by MLCommons.

Understand alignment-helpfulness trade-offs

While overall model safety should keep improving as models advance, some trade-off between model helpfulness and model alignment is likely unavoidable. That's because any prediction–Is this content aligned? Is this content unaligned?–carries at least some risk of applying content policies falsely (i.e., false positives and false negatives.) These errors will necessarily mean that a model will either be more aligned and less helpful or less aligned and more helpful.

To illustrate: Consider a content policy against assistance with scams. If a user submits a prompt for "How does a ponzi scheme operate?" the model can either refuse to substantively answer (arguably the most aligned, least helpful option) or provide a complete, detailed answer (arguably the most helpful, least aligned option). Consider the same evaluation, but with the prompt *"How to protect yourself from identity theft."*

As the model's rate of identifying and stopping unaligned content grows, its likelihood of falsely stopping aligned content–and thereby reducing its overall helpfulness–grows in tandem. In other words, you'll need to look elsewhere to learn about stopping identity theft. Turning down the dial–so that more unaligned content gets through–will likely have the knock-on effect of increasing the likelihood that the model generates helpful content. You'll learn about protecting your identity from thieves.

Avoiding alignment-helpfulness trade-offs is probably impossible. But developers should exercise discretion about how to weigh the benefits of alignment and helpfulness for their specific use case and audience. We look forward to exploring more ways to give developers greater control over this important aspect of model building.

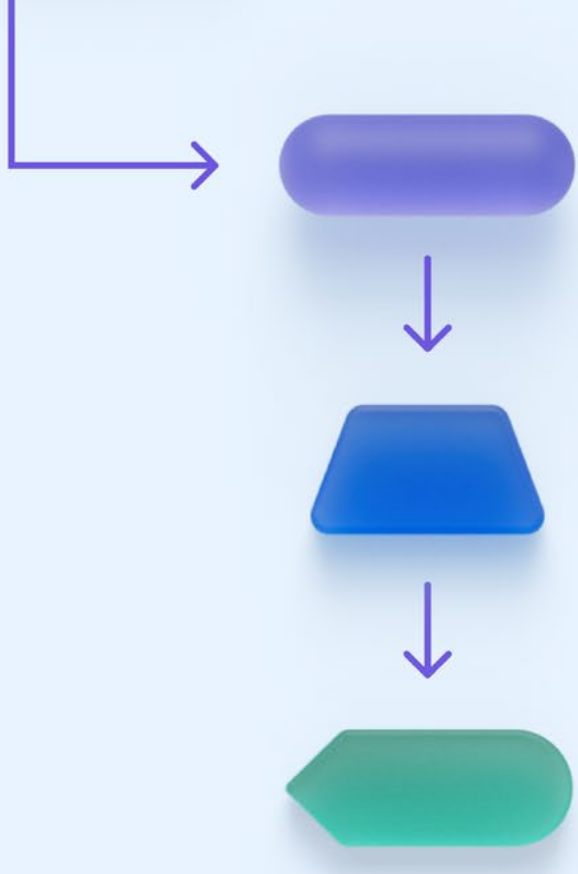
2

Model-level alignment

Product-specific fine-tuning enables developers to leverage pre-trained models or models with some fine-tuning for a specific task requiring only limited data and resources. Even with initial fine-tuning performed by Meta, developers can further train the model with domain-specific datasets to improve quality on their defined use case. Fine-tuning adapts the model to domain- or application-specific requirements and introduces additional layers of safety mitigations. Examples of fine-tuning for a pre-trained LLM include:

- **Text summarization:** By using a pre-trained language model, the model can be fine-tuned on a dataset that includes pairs of long-form documents and corresponding summaries. This fine-tuned model can then generate concise summaries for new documents.
- **Question answering:** Fine-tuning a language model on a Q&A dataset such as SQuAD (Stanford Question Answering Dataset) allows the model to learn how to answer questions based on a given context paragraph. The fine-tuned model can then be used to answer questions on various topics.
- **Sentiment analysis:** A model can be fine-tuned on a dataset of labeled text reviews (positive or negative sentiment) to recognize sentiment and perform analysis to understand user satisfaction. By training the model on this task-specific dataset, it can learn to predict sentiment in text accurately.

These examples showcase how fine-tuning an LLM can be used to specialize the model's capabilities for specific use cases, improving its performance and making it more suitable for specific applications. The choice of the foundation model and the task-specific dataset plays a crucial role in achieving the desired results.



The responsible fine-tuning flow

Here are the general steps needed to responsibly fine-tune an LLM for alignment, guided at a high level by Meta's Responsible AI framework:

1. **Prepare data**
2. **Train the model**
3. **Evaluate and improve performance**

Step 1: Prepare data

Developing downstream applications of LLMs begins with taking steps to consider the potential limitations, privacy implications, and representativeness of data for a specific use case. Begin by preparing and preprocessing a clean dataset that is representative of the target domain. This involves tokenizing the text, handling special characters, removing unnecessary information, and splitting the dataset into training, validation, and testing sets. This step may also involve ensuring that data are representative of the end users in the deployment context, for instance, by ensuring there are enough examples from relevant languages if you plan to deploy your product in a non-English speaking market. Representativeness of data is dependent on the use case and should be assessed accordingly.

When fine-tuning for a specific use case it can be beneficial to examine training data for biases, such as gender, racial, linguistic, cultural or other biases.

Understanding these patterns is important but it may not always be optimal to filter out all problematic content in training data due to the unintended consequences this filtering may have on subsequent performance and safety mitigations, such as prompt engineering. Instead of removing data, focusing on the representativeness of the data can help prevent a fine-tuned model from perpetuating biases in its generated outputs; what is considered representative will depend on the specific context in which a product is deployed. Developers should also pay attention to how human feedback and annotation of data may further polarize a fine-tuned model with respect to subjective opinions, and take steps to prevent injecting bias in annotation guidelines and to mitigate the effect of annotators' bias. Resources on this topic include:

- Don't Blame the Annotator: Bias Already Starts in the Annotation Instructions
- Annotators with Attitudes: How Annotator Beliefs And Identities Bias Toxic Language Detection

There are several other risks to consider, such as overfitting, privacy, and security. To mitigate these risks, carefully design the fine-tuning process by curating a high-quality dataset that is representative of your use case, conduct rigorous evaluations, and test your fine-tuned model's potential use via red teaming (covered in step four - *Evaluate and improve performance*).

Step 2: Train the model

Fine-tuning involves training the model for a limited number of iterations. Once a pre-trained model is loaded in the environment for fine-tuning, the training process involves setting up hyperparameters like epochs, batch size, and learning rate. The data are passed through the model, loss is computed, and weights are updated through backpropagation. The training progress is monitored using a validation set, and hyperparameters are adjusted as necessary.

Fine-tuning an LLM for safety can involve a number of techniques including:

- **Supervised Fine-Tuning (SFT):** Supervised fine-tuning using data annotated across helpfulness and safety.
- **Reinforcement Learning from Human Feedback (RLHF) or AI Feedback (RLAIF):** Training safety and helpfulness reward models to support RLHF techniques iteratively improves models and makes them more robust to jailbreaking techniques.
- **Targeted Safety Context Distillation:** Context distillation for safety helps the model associate adversarial prompts with safe responses by prefixing a safe pre-prompt such as "You are a safe and responsible assistant" to the adversarial prompt, followed by fine-tuning on new outputs.

Reinforcement Learning from Human Feedback (RLHF)

To align the output of LLMs with user expectations and values, one approach that developers should consider is implementing Reinforcement Learning from Human Feedback (RLHF) mechanisms. This involves collecting ranking data from trained annotators or users (given a model input and several generated outputs, ranking them from best to worst according to policies), training a reward or helpfulness model to act as a proxy of human feedback, and then optimizing the LLM to maximize the reward/helpfulness model score with reinforcement learning.

Reinforcement Learning from AI Feedback (RLAIF)

Reward models can also be improved and tailored to specific policies by using Reinforcement Learning from AI Feedback (RLAIF). The fine-tuned LLM itself can be used to create synthetic ranking data for reward model training. Given a model input, response pairs and relevant guidelines, the LLM predicts which response would best follow the guidelines. The synthetic reward modeling data are then used to augment the reward model's training data.

Step 3: Evaluate and improve performance

The final stage is to evaluate the fine-tuned model on a test set to measure its performance on the specific task and against safety benchmarks, according to the use case. This includes analyzing the model's strengths and weaknesses based on evaluation results, gathering more data to further enhance performance and safety, and iterating until satisfied with the model's performance using holdout test datasets.

There are many complementary types of evaluations that are useful for measuring risks in models, including automatic benchmarks, manual annotations by human raters, and evaluations using an LLM itself as a rater. The Holistic Evaluation of Language Models discusses some of the most commonly used automatic benchmarks. As the industry matures, we are excited for evaluation platforms to emerge to help drive safety standardization, such as through the MLCommons AI Safety working group. Evaluation strategies and processes to improve performance can include:

- **Automatic evaluation** leverages automatic benchmarks and classifiers to judge the output with respect to a specific category of risk.
- **Manual evaluation** leverages human annotators or subject matter experts to judge the model's output.
- **Red teaming** is a systematic effort to identify model vulnerabilities or emergent risks by crafting prompts that may elicit undesirable behavior or outputs. This type of manipulation of the model can be used to test safeguards and attempts to "jailbreak" the model.

Red teaming best practices

Red teams should adopt systematic approaches to testing and measurement, while estimating real-world behaviors and threat vectors to the extent possible.

- **Diversity:** Red teams should include a diverse set of people from a range of professional backgrounds that are representative of a broad group of potential users and demographics. Red teams can be composed of internal employees, experts, or community members.
- **Subject matter expertise:** Subject matter experts should judge model responses based on their familiarity with the identified risk categories and label responses that fall under each category.
- **Regular testing:** The model should undergo regular testing to determine whether or not mitigations against attacks are effective. This requires some form of automated evaluation, either with human labeling, which can be expensive, or with classifiers trained to recognize responses that fall under the risk categories.

3

Privacy adversarial attacks

Additional privacy protections should be considered when releasing the product, to test whether bad actors may be able to improperly extract information. A privacy adversarial attack is a method where attackers can exfiltrate data from a model. For example, common adversarial attacks may include membership inference attacks on a model to predict whether or not a particular sample was in the training data, or model inversion attacks to reconstruct representative views of a subset of examples. Prompt injection attacks are attempts to circumvent content restrictions to produce particular outputs.

A red team privacy adversarial attack conducted by a company may be able to demonstrate the feasibility of such attacks. In scenarios where companies fine-tune models using personal data (pursuant to applicable privacy laws), they should consider testing the outputs to see if the model memorized particular data. This approach may be especially useful for testing models that are intended to be deployed as AI assistants or agents.

System-level alignment

Without proper safeguards at the input and output levels, it is hard to ensure that the model will respond properly to adversarial inputs and will be protected from efforts to circumvent content policies and safeguard measures ("jailbreaking"). Mitigations at the output level can also act as a safeguard against generating high-risk or policy-violating content. Enforcement of content policies can be managed through automated systems and manual analysis of samples and reports. Automated systems may include machine learning and rule-based classifiers for filtering prompt inputs or system outputs. Usage or consequence policies may be defined for when users repeatedly violate those policies.

Enforcement of content policies can be managed through automated systems and manual analysis of samples and reports.

Automated systems may include machine learning and rule-based classifiers for filtering prompt inputs or system outputs. Usage or consequence policies may be defined for when users repeatedly violate those policies.

Mitigating risks at the input level

The input refers to the information provided by the user and passed to the system. The developer does not control what the user inputs. Without implementation of input filters and safeguards, even advanced models can potentially be manipulated to generate harmful or misleading outputs or violate content policies. Although safeguards to protect privacy and prevent potential harm can be developed by tuning the model, it should be expected that even after rigorous design and testing, those safeguards will not have perfect performance and may be subverted. Additional safeguards include direct filtering and engineering of the inputs. For these to be effective, model inputs must be well-formatted. These approaches include:

- **Prompt filters:** Even when inputs may not violate content policies, the model may produce problematic engagements or outputs. In these cases, it may be appropriate to filter, block, and hard code responses for some inputs until the model can respond in the intended way. This tactic may come with tradeoffs to the user's experience and agency in engaging with the system. Thus, the safety benefits of such restrictions or modifications should be weighed against those costs, until more robust solutions are developed.
- **Prompt engineering:** Direct modifications of the user inputs are an option for guiding the model behavior and encouraging responsible outputs, by including contextual information or constraints in the prompts to establish background knowledge and guidelines while generating the output. Modifications may be done in a variety of ways, such as with automated identification and categorization, assistance of the LLM itself, or rules engines. These can help improve the user experience by creating more diversity and expressiveness from the model. For example, prompt engineering can be leveraged to direct the model to include more diverse references or apply a certain tone or point of view. Prompt engineering rules may be hard coded or probabilistic.

Alongside prompts, it might be beneficial to provide instructive sample inputs and outputs that illustrate the desired responsible behavior.

Mitigating risks at the output level

Based on the downstream use case, you can apply several approaches for detecting and filtering the generated output of models for problematic or policy-violating content. Here are some considerations and best practices for filtering outputs. Any output filter mitigation should include all languages that are used in the region where your product is available.

- **Blocklists:** One of the easiest ways to prevent the generation of high-risk content is to compile a list of all the phrases that your model should not, under any circumstances, be permitted to include in a response. Many words are easily identifiable as problematic; slurs, for example, are typically offensive no matter their context. While blocklists are attractive for their simplicity, they may unreasonably restrict the usage of your model. Words often have context-dependent meanings, and terms that could be sexually suggestive, for example, may also be used in medical contexts. Content policies will help articulate the specifics between permitted and prohibited topics to users.
- **Classifiers:** The more effective, but also more difficult, approach is to develop classifiers that detect and filter outputs based on the meaning conveyed by the words chosen. Classifiers, when properly trained on known examples of a particular sentiment or type of semantic content, can become highly effective at identifying novel instances in which that sentiment or meaning is expressed.

4

Evaluate effectiveness

While prompt filtering and engineering are critical safety mitigations, it's important to monitor effectiveness and avoid unintended consequences. Some best practices include:

- **Test for unintended outcomes.** Take caution that prompt engineering doesn't inadvertently create other issues. Test end-to-end performance after any prompt engineering to ensure desired behavior.
- **Evaluate effectiveness of safeguards.** Many publicly available datasets offer collections of prompts that are designed to benchmark against specific concerns when used as inputs. After model responses are collected, they can be evaluated by using standardized metrics.
- **Adjust for different languages.** Prompt filtering and engineering mitigations should include all languages that are used in the region where your product is available; the effectiveness of these mitigations may be dependent on linguistic and community-level nuances.

Build transparency and reporting mechanisms in user interactions

Releasing an LLM-powered feature for users to interact with can reveal new use cases as well as new concerns. User interactions can provide critical feedback, which can be used for reinforcement learning (discussed in a previous section). This is also an opportunity to provide appropriate notice, transparency, and control to users, which can lead to greater satisfaction and trust in the feature.

Feedback & reporting mechanisms

Facilitating user interaction with appropriate feedback or reporting mechanisms is key to ensuring quality output. Feedback mechanisms can be as simple as positive or negative (thumbs up or thumbs down), and tailoring feedback to the types of issues that may be foreseeable based on a company's use case (for example, AI assistants) can enhance the quality of feedback. This feedback can be used by developers to improve the model in more targeted ways. Providing an option for freeform feedback within a reporting mechanism can also reveal new or unanticipated concerns raised by users. Furthermore, users can identify and highlight errors, unsafe behaviors, or suboptimal actions that the model might not recognize on its own. Developers can further train the model with this feedback to improve performance and avoid repeating mistakes. Product

developers should review feedback by monitoring the rate that users report model outputs and by manually reviewing those reports and selected samples of model outputs.

Transparency & control best practices

To ensure high-quality feedback and provide end users with notice and choice about their interactions with your AI assets, developers should consider the following practices for user interactions:

- **Transparency:** Developers should consider ways to provide transparency to end users regarding potential risks and limitations of the system prior to or at the time of user interaction. For instance, notice to users that they are interacting with an AI-powered chatbot may increasingly be required in certain markets, and is a best practice to address concerns that may be related to false or incorrect information. Developers should also ensure that the use of any third party tools integrated with an LLM (eg, search) to generate specific content or supplement the capabilities of the LLM is clear to end users. Developers should neither claim nor imply that an AI agent is human, especially when building and deploying anthropomorphized interfaces. Context, intent, sensitivity and likelihood to deceive are additional critical factors in ascertaining when and how to be transparent. Work with your appropriate advisors to determine the types of transparency that should be provided to users, including whether users should be informed that their responses may be used to fine-tune a model. Developers should also consider the use of system cards to provide insight into their AI system's underlying architecture and explain how a particular AI experience is produced. Further best practices are outlined in the Partnership on AI's Responsible Practices for Synthetic Media.
- **Control mechanisms:** Additional controls could include giving users the option to customize the outputs generated by an LLM. For example, a user could select or reject outputs from a list of multiple options. Offering editing capabilities can also enhance a user's sense of agency over outputs, and developers should consider education flows that can set a user up for success, such as offering prompt suggestions or explanations of how to improve an output.

Responsibility considerations for LLM Capabilities

LLMs are becoming more capable as the technology continues to advance. New capabilities can take various forms, from supporting a new modality, more complex interactions or being trained for a specific skill. Each step forward opens the door for developers to build new use cases and achieve increasingly sophisticated tasks.

Building with these new capabilities requires specific considerations in addition to the best practices mentioned above that generally apply across all Generative AI use cases. Developers should get familiar with the information below if their application is related or leverages capabilities mentioned below.

Image reasoning

Vision-Language models are models that were trained on text and image simultaneously. They have the ability to take as input both text and image modalities. Their image reasoning capability can be used for various tasks like visual questions answering.

Developers should refer to the best practices outlined in the "mitigating risks at the input level" section of the Responsible Use Guide to understand how to responsibly implement image input filters and safeguards to limit harmful text output generation. Employing approaches such as prompt filtering and prompt engineering can help protect privacy and prevent potential harm. It is important to note that image inputs add an additional level of complexity and difficulty for mitigations such as prompt filtering.

Developers are responsible for deploying additional filters to prevent the upload of illegal images and should be used as appropriate to ensure compliance with all applicable laws and regulations.

In addition to the best practices outlined previously, specific attention should be paid to risks emerging from the potential processing of people in images. Developers should restrict the input of images and queries that would disclose private or sensitive information about individuals, including information about individuals' identity, health, or demographic information without obtaining the right to do so in accordance with applicable law.

Enabling a new input modality also introduces new attack vectors and potential prompt injections that may be contained in the image. For example, malicious users may embed text instructions or make indirect references by rephrasing a restricted request using multiple modalities. It is recommended to use additional specialized system safeguards such as Llama Guard-vision for input/output filtering to meaningfully reduce risks and always assess the risk profile of application in their usage context.

Tool calls

LLMs are foundational models that can be integrated in complex workflows to achieve sophisticated tasks. In order to interact with other components, LLMs are often fine-tuned to respond in a structured format (e.g. JSON) to make the right API call based on the user input.

Example

The Following user prompt “what will the weather be like tomorrow” would trigger the LLM to make a call to a weather or search provider to get the most up-to-date information on the weather.

Llama 3.2 instruct models have been fine-tuned to support specific tool calls, such as search, or generating code to be executed by the hosting environment. Please refer to our paper and relevant model cards to learn about the supported calls and the safety mitigations we implemented as part of the model development.

In addition, developers should consider the following before deploying such applications :

1. Deployment of appropriate system safety solutions to mitigate risks introduced by the tool

Developers are expected to deploy system safeguards relevant to the tools they integrate. As a starting point, we recommend integrating content moderators such as Llama Guard to validate that no violating text content was introduced by the third party tool before the response is output to the end user. However, if developers intend for their system to call tools that generate modalities other than text, then additional system level safeguards beyond Llama Guard will also likely be required.

2. Mitigation of security risks

LLM-tool integrations introduce security risks in a number of ways, including: the tool can send a poisoned request to the LLM, and the LLM can generate malicious queries for the tool. We recommend integrating safeguards like Prompt Guard to detect direct or indirect LLM-jailbreak attempts, or Llama Guard to limit the risk of your hosting environment executing malicious code generated by the LLM in the case of code-interpreter tool use.

Just like in standard software development, developers are responsible for the integration of the LLM with the tools and services of their choice. They should define a clear policy for their use case and assess the integrity of the third party services they plan to use to be aware of any safety and security limitations when using this capability. Developers should ensure they use reputable and responsible third party tool providers and be transparent where tools are providing or supporting the generation of outputs within their system, through the use of watermarks or other transparency notices.

Multilinguality

Expanding LLM's abilities to support as many languages as possible is key to an open approach and to ensure Generative AI technology benefits everyone.

When using LLMs in a given language, developers must ensure this language is fully supported by the LLM, both in terms of performance and safety. The information should be available in the Model Card or Acceptable Use Policy of the LLM.

Note that LLMs are usually trained on a vast amount of data, which may include different languages. As a result an LLM may be able to output text in a given language and it may appear as if the language was supported, but the LLM might not have been optimized nor properly evaluated for safety in that language. Developers should carefully assess supported languages and limitations or risks associated with use in languages other than those supported to ensure the appropriate level of safety for their end users.

Languages might be represented in training data to various extent, and the richness of each language, their nuances and locale specificities might not be well reflected. Developers should conduct extensive testing to ensure their language cultural references and values are well reflected.

For example, in an effort to support multilingual expansion, we optimized Llama Guard, our system-safety text content safeguard, to support new languages beyond English. Adding language filters to your system is also a good practice to prevent access to unsupported languages that might now be adequately safety tuned for a given use case.

Coding

LLM Coding capabilities have the potential to make workflows faster and more efficient for current developers and lower the barrier to entry for people who are learning to code, and also to be used as a productivity and educational tool to help programmers write more robust, well-documented software.

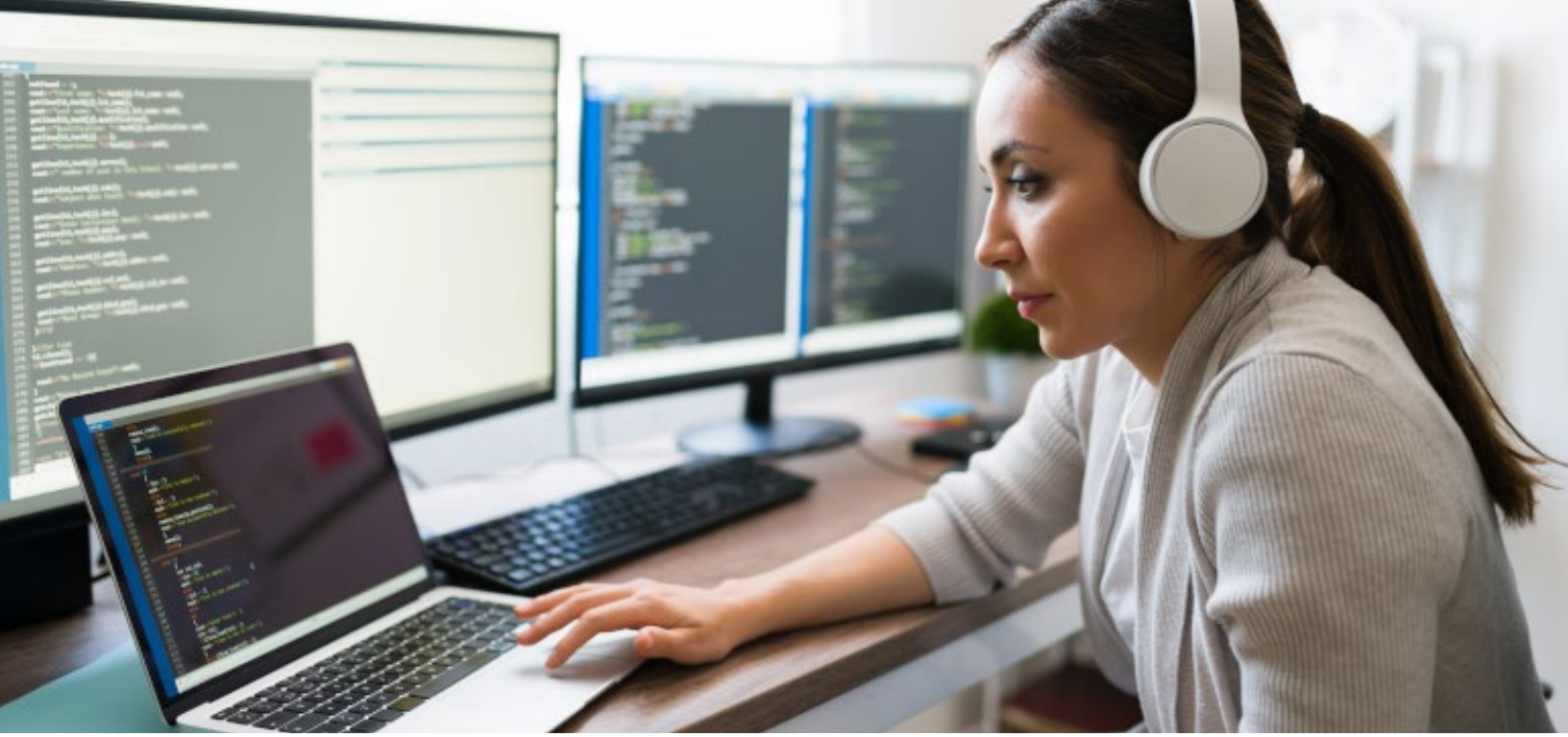
Developers should refer to the Responsible Use Guide, but retain responsibility for considering and adhering to code-specific best practices when building on top of LLMs in line with their specific use case.

Define content policies for use case

- In the code domain, models should avoid producing malware, datasetsviruses, or other malicious code. Developers should consider how bad actors prompt the model to produce these results and are ultimately responsible for exploring and implementing mitigations and safeguards that make sense for their specific use case and environment.

Evaluations & benchmarks

- Code models should be evaluated against code-specific benchmarks, such as CyberSecEval.



Safety studies & fine-tuning considerations

- The data should be representative of the end users' requirements. For example, if the model is meant for Javascript generation, the dataset chosen to fine-tune with should be Javascript-focused. Developers should also consider examining and placing restrictions on any potentially malicious or nefarious code in the data.
- Developers should ensure the security and robustness qualities of the training code dataset matches the security requirements of the output and the systems where the output code will be integrated based on a specific use case.
- Developers should perform safety studies on code-specific areas such as intentional malware generation and the unintentional introduction of vulnerable code. Working with red-teaming domain experts can help developers evaluate the model's capacity to lower the bar for writing malicious code when the prompt intent is clear and the output goes beyond resources already publicly available on the Internet and other publicly available sources.
- If the model's output will be used in production/non-test systems, developers should ensure the code that the model is trained on is free of relevant security vulnerabilities. Developers and end-users that use the model as an assistant for software development should continue to follow security best practices.

System safeguards

Developers should deploy coding specific safeguards. For example, Code Shield can be used as an output filter to limit the risk of insecure code being generated and integrated in production code bases. Llama Guard can be used to limit the risk of LLMs providing helpful responses to cyber attack prompts.

Combining the components of responsible generative AI

Each stage of model development presents opportunities to enhance the safety of your AI feature. However, it's crucial to acknowledge the interconnectedness of these stages and how the decisions made at each stage can impact others. Building a responsible AI ecosystem requires ongoing efforts to refine each component and ensure they work together effectively.

Here are some key considerations for implementing these components in unison:

- **Holistic optimization.** Although each component has a specific role and optimization goal, components are not isolated entities. Over-optimization of one component without considering its interaction with others can lead to suboptimal outcomes. For instance, over-filtering training data for safety might make later fine-tuning less effective, as the model may not recognize and handle unsafe content appropriately. This is why different layers of safety mitigations throughout the development lifecycle are critical for creating high-performing, responsible products.
- **Alignment of objectives at each stage of development.** To yield a product that is optimized for your target use cases, it's essential to have a consistent set of goals and outcomes that guide each stage of the process. From the data-collection stage to user feedback, be sure to keep your overall goal in mind.
- **Standardizing processes for learning from feedback/errors.** Embracing an iterative model-development mindset is crucial. Establish a well-defined process for incorporating new learnings into subsequent model training. This process should include consistent feedback analysis, prioritization of identified issues, and systematic application of learnings in the next iteration of model training.

The field of generative AI is complex, ever-evolving, and full of potential, but it's not without risks. The key to unlocking its benefits while mitigating the downsides is responsible AI practice. This practice starts with understanding the complexities of the technology, the potential impacts on users and society, and the importance of continuously striving for improvement.

By embracing the principles of transparency, accountability and user empowerment, as well as having a commitment to ongoing learning and improvement, you can ensure that your AI feature is not only innovative and useful but also responsible and respectful. We hope this guide serves as a valuable tool in your journey toward responsible AI practice.