Meta

Meta Llama

Responsible Use Guide

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

Contents

Meta

Meta is committed to open science because we believe that a vibrant AI-innovation ecosystem will push the frontiers of scientific discovery and potentially revolutionize a wide array of sectors from education to agriculture, and climate management to cybersecurity.

We believe that the power of AI will be harnessed to address global challenges, and unlocking that power responsibly will require democratization of access and collaboration on risk management. We want to empower developers in every industry on a global scale to drive breakthroughs, create new products and solutions, and benefit from accelerations in technological advancement and economic growth.

Meta has open sourced code and datasets for machine translation, computer vision, and fairness evaluation, while contributing to the infrastructure of the AI-developer community with tools like PyTorch, ONNX, Glow, and Detectron. In the past, we have also made our cutting-edge large language models (LLMs) Llama 1 and OPT-175B available to the scientific community through research releases which have spurred research in model efficiency, medicine, and conversational safety studies on evaluation methods, de-biasing techniques, and sources of hallucinations in LLMs.

We also took an important step toward advancing access and opportunity in the creation of AI-powered products and experiences with the launch of Meta Llama 2. The open release of these models to the research and business community laid the foundation for the next wave of community-driven innovation in generative AI. We've seen an incredible response thus far with millions of download requests in the time since its release.

Democratization of access will put these models in more people's hands, which we believe is the right path to ensure that this technology will benefit the world at large. We take our commitment to building responsible AI seriously, cognizant of the potential privacy and content-related risks, as well as societal impacts.

Meta is proud to have supported the Llama 2 developer community by building state-of-the-art responsibility tooling that makes it easier than ever to build and release models responsibly. Learn more about the open source tools we share with developers to help them build responsibly.

We're also excited to release an early look at the next generation of Llama, Meta Llama 3 which, like Llama 2, is licensed for commercial use. This release of Llama 3 features both 8B and 70B pretrained and instruct fine-tuned versions to help support a broad range of application environments.

We envision Llama models as part of a broader system that puts the developer in the driver seat.

Llama models will serve as the foundational piece of a complex system that developers design with their unique end goals in mind. As part of this system centric approach, and to support responsible deployment of these models, we have updates to our open trust and safety project including a Meta Llama Guard 2 model that supports a broader taxonomy for input/output prompt filtering.

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 at various points in the system. 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 & 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. With our release of Llama 3 paired with Llama Guard 2, we are beginning to extend this vision of a layered approach to safety to our open models as well.

As discussed in our research paper on Llama 2, 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 model 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

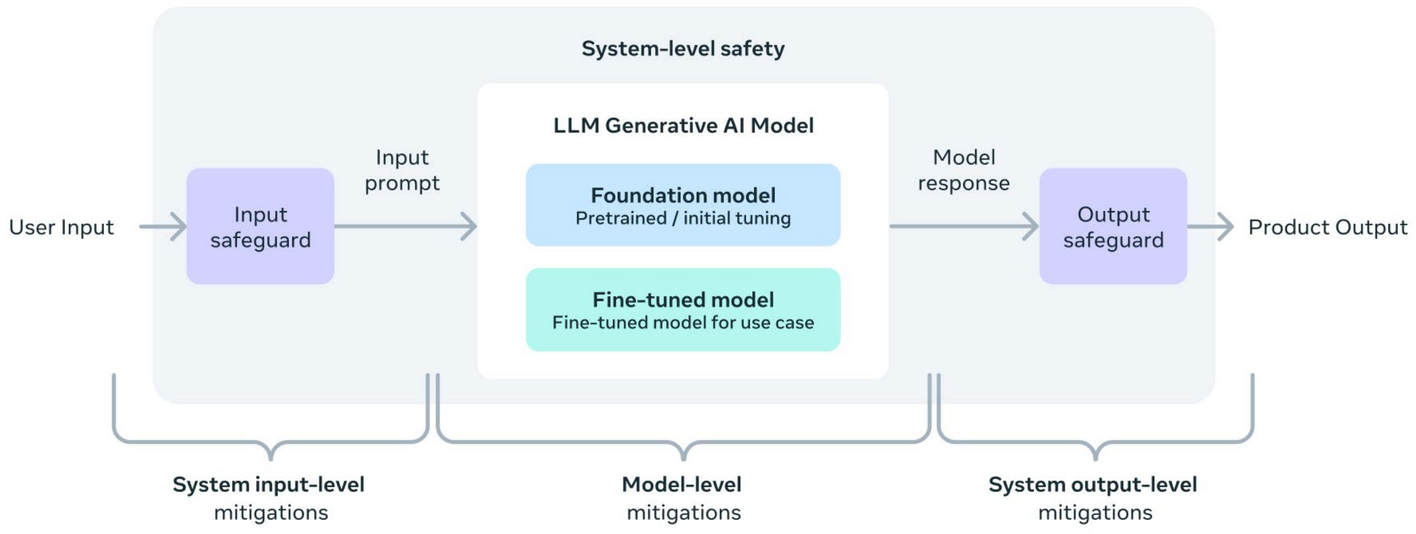
A foundation model is a general purpose AI technology whereas an LLM-powered product has a defined use case and performs specific tasks to enable an intended use or capability through a user interface, sometimes embedded in products. An LLM-powered system encompasses both the foundation model and accompanying 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 explore these options.

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



Development of the foundation model

Meta Llama 3, like Llama 2, is licensed for commercial use. This release of Llama 3 features both 8B and 70B pretrained and instruct fine-tuned versions to help support a broad range of application environments. This next generation of Llama demonstrates state-of-the-art performance on a wide range of industry benchmarks and offers new capabilities, including improved reasoning. With the developer in mind, and in support of our longstanding open approach, we wanted to put Llama 3 in the hands of the community as soon as possible to enable early development and kickstart this next wave of innovation.

In addition to performing a variety of pretraining data-level investigations to help understand the potential capabilities and limitations of our models, we applied considerable safety mitigations to the fine-tuned versions of the model through supervised fine-tuning, reinforcement learning from human feedback (RLHF), and iterative red teaming (these steps are covered further in the section - *Fine-tune for product*).

More information on Llama 3 model architecture and parameters and pretrained evaluations are contained in the model card. The model card also provides information about the capabilities and limitations of the models.

During pretraining, a model builds its understanding of the statistical patterns across the sample of human language contained in its training data.

The training datasets for Llama are sourced from a broad set of diverse, publicly available online data. This training corpus is mostly English, which is consistent with the current, intended use of the model. For each dataset used in training, we followed Meta's standard privacy review processes. And for our pretraining data we made an effort to remove data from certain sources known to contain a high volume of personal information about private individuals. After pretraining, 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 pretrained model, we recommend tuning it by using the techniques described in the next section to reduce the likelihood that the model will generate 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. We also recommend using Llama Guard 2 for enhanced safety performance.

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

1 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 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.

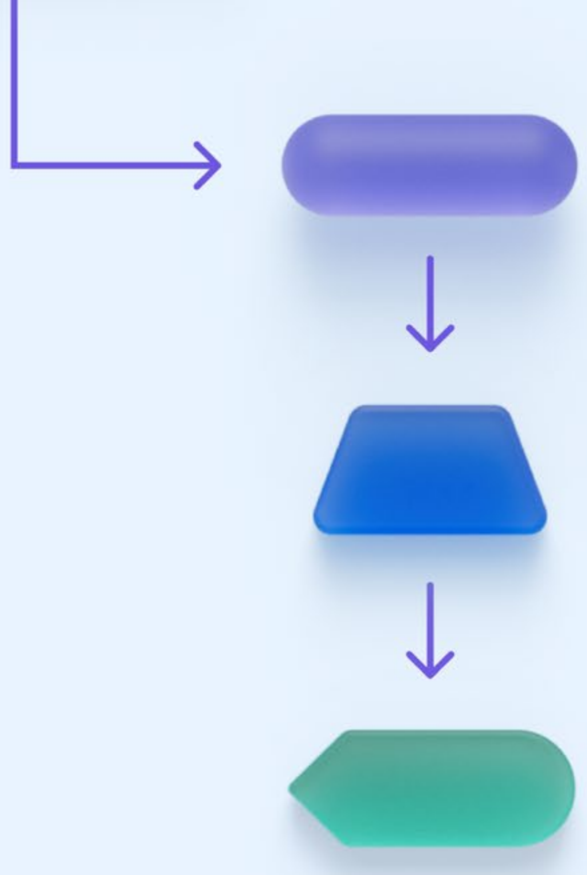
2

Model-level alignment

Product-specific fine-tuning enables developers to leverage pretrained 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 pretrained LLM include:

- **Text summarization:** By using a pretrained 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 pretrained 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, many of which the research paper on Llama 2 describes in greater depth. These techniques can include:

- **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 preprompt 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.

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. Llama was trained primarily on data in English, in accordance with its intended use, so it is critical to carefully evaluate any mitigations in other languages.

4

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 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.

Resources for developers

There is a value chain emerging to support the responsible training and use of LLMs, which we believe will be advanced through more open releases and sharing of best practices, tools, and benchmarking. A growing number of researchers, platforms, companies, and developer communities are contributing to this ecosystem. We expect more tools for the responsible development of LLM to become available over time and are committed to fostering more open exchange of safety research and tools to support developers.

It is critical to remain aware of the latest versions of models and use the most current version to get the best results.

Our partnership to make Llama available on the Azure Model Catalog will enable developers using Microsoft Azure to leverage their cloud-native tools for content filtering and safety features. Below, we provide a few notable hubs and implementation resources for developers, but this list is not exhaustive. Microsoft also offers a repository of Responsible AI Resources.

Filters and classifiers:

- Meta Llama Guard and other solutions
- Content-filtering systems from Azure, supporting a range of languages: learn.microsoft.com/en-us/azure/cognitive-services/content-safety/overview
- Filter lists for generation of problematic words: github.com/LDNOOBW/naughty-words-js
- Recipes for safety in open-domain Chatbots, including a sensitive topics classifier: parl.ai/projects/safety_recipes/

Platforms for tools and evaluations:

- MLCommons AI Safety (v0.5): mlcommons.org/2024/04/mlc-aisafety-v0-5-poc/
- Meta Llama Cybersecurity evaluations
- Benchmarking of LLMs by Stanford's Center for Research on Foundation Models, HELM: crfm.stanford.edu/helm/latest/
- EleutherAI LLM Evaluation Harness: github.com/EleutherAI/lm-evaluation-harness
- Hugging Face Hub which hosts open source models, datasets, and is a space for developers to share safeguards and access benchmarking information: huggingface.co/docs/hub/index

Reporting resources

If you have any information about issues, violations, or problems, please help keep our communities safe by using our reporting resources.

- Reporting issues with the model: [github.com/facebookresearch/llama]()
- Reporting risky content generated by the model: [developers.facebook.com/llama_output_feedback]()
- Reporting bugs and security concerns: [facebook.com/whitehat/info]()
- Reporting violations of the Acceptable Use Policy or unlicensed uses of Llama: [LlamaUseReport@meta.com]()

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.

Introducing Code Llama

Code Llama is a set of large language models for code based on Llama 2 providing strong code generation and infilling capabilities, support for large input contexts, and zero-shot instruction following ability for programming tasks, as well as state-of-the-art base models for fine-tuning. We provide multiple flavors to cover a wide range of applications: foundation code models (Code Llama), Python specializations (Code Llama - Python), and instruction-following models (Code Llama - Instruct) with 7B, 13B, 34B and 70B parameters each. All models are trained on sequences of 16k tokens and show improvements on inputs with up to 100k tokens. The 7B and 13B Code Llama and Code Llama - Instruct variants support infilling based on surrounding content. Code Llama was developed by training Llama 2 on publicly available code and natural language datasets related to code.

Building AI models responsibly is important to us, and we undertook a variety of safety measures before releasing Code Llama, including many of the measures used for Llama 2. Before releasing the 7B, 13B, and 34B variants in August 2023, we asked cybersecurity and malware development experts to evaluate the Code Llama model's capacity for enabling malware development by otherwise unskilled adversaries. We subsequently expanded our evaluation suite to include Meta's CyberSecEval, which evaluates an LLM's capacity for producing code with known insecure patterns or complying with a cyber attacker's request. Measuring the model's response to malicious requests allowed us to implement mitigations which make Code Llama 70B safer than other available models, while remaining helpful. For detailed information on model training, architecture and parameters, evaluations, responsible AI and safety refer to our research paper and Responsible Use Guide.

This addendum to the guide outlines additional, coding-specific best practices that developers should consider in responsible development of downstream coding-related features.

Code Llama potential use cases

Code Llama has two broad use case categories:

1. **Foundation model use case**: Train on more data to create variants of Code Llama, e.g, add other programming languages (C++, Java), increase context length, reduce latency by quantization of model, etc.

2. **Instruction model use case**: Add more instruction data to improve the model's ability to follow instructions, extend capability to understand non-English instructions, create standalone code bots and integrate into existing 3rd party products, etc.

Both the foundation and instruction Code Llama models are not designed to be used as a large language model for general purpose - for such scenarios refer to Llama 2.

1

Foundation model use case

This model could be used for further research exploration on specialized foundation large language models for programming. When fine-tuning Code Llama, we refer users to the Responsible Use Guide for Llama 2, which provides essential guidance on responsible development of downstream models, including on (i) defining content policies and mitigations; (ii) preparing data; (iii) fine-tuning the model; (iv) evaluating and improving performance; (v) addressing input- and output-level risks; and (vi) building transparency and reporting mechanisms in user interactions.

Additionally, developers should consider code-specific best practices when building on top of Code Llama in line with their specific use case.

Define content policies for use case

- A content policy defines what content is allowable based on the intended use and deployment context. This may also outline limitations on producing potentially problematic content. In the code domain, models should avoid producing malware, virus, or malicious code. Developers should consider how bad actors prompt the model to produce these results.
- These policies will dictate the data needed, annotation requirement, and goals of safety fine-tuning. They may also be applied in input- and output-level safeguards as additional safety mechanisms.

Evaluations & benchmarks

- Models should be evaluated against their intended use and end user requirements. Specifically, code models should be evaluated against code-specific benchmarks. As a resource, you can find various benchmarks on Papers with Code: Code Generation Benchmarks.
- We recommend evaluating the cybersecurity safety of coding models with the CyberSecEval (GitHub) which was released as part of our open trust and safety tools and evaluations project.
- Non code-specific safety evaluations are also recommended, for example, code models can be evaluated on benchmarks such as TruthfulQA, ToxiGen and BOLD.

Red teaming & 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.
- Developers and end-users that use the model as an assistant for software development should be aware of the model's overall language safety. Performing safety studies and comparing results to representative benchmarks can identify particular categories of content risk. To mitigate those risks, collect relevant fine-tuning data that is not within the test set, and fine-tune the model by controlling for higher measured safety while maintaining helpfulness.
- If the model's output will be used in production 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-level safeguards

- As for other LLM use cases, we recommend adding appropriate system-level safeguards in addition to the model-level built-in alignment.
- As part of our open trust and safety tools, we released Code Shield, an insecure code detector that can be used at inference to prevent bad coding practices. Visit Code Shield Github for more information on how to integrate it into your application.

When using Code Llama in particular, it is important to keep in mind that Code Llama is specialized for code-related tasks and may not be appropriate as a foundation model for other task families, e.g, general language model. We note that for downstream tasks where the Python programming language is more relevant, it may be more appropriate to use our Code Llama - Python model variant. Code Llama and Code Llama - Python are not trained with instruction data hence are not designed to follow instruction in natural language. Any use of these models to perform general natural language tasks is not recommended to avoid potential misuse of the models.

2

Instruction model use case

This model could be used for further applied research and testing of specialized large language models for programming. Code Llama - Instruct has the ability to understand instructions in natural language. When using Code Llama for code generation tasks, we recommend developers use our Code Llama - Instruct variants, which have been fine-tuned to generate helpful and safe answers to users. Consult the full Responsible Use Guide for best practices with regards to generally addressing input- and output-level risks and building transparency and reporting mechanisms in user interactions, as relevant to a specific use case.

For both use cases, users must abide by our Acceptable Use Policy.

Reporting resources for developers

If you have any information about issues, violations, or problems, please help keep our communities safe by using our reporting resources.

- Reporting issues with the model via our GitHub repo
- Reporting risky content generated by the model via our Developer portal
- Reporting bugs and security concerns via our Bug Bounty