API Usage Guidelines

1. Introduction and Overview

4Five Labs's API can support a range of applications like classification and question-answering. While we expect that the API will mostly create benefits for customers and end users, it also creates safety risks that are important to characterize and mitigate.

This document begins with general context on how to think about safe use of systems like the API, then enumerates several specific safety issues to consider, provides general guidance on risk mitigation, and provides more detailed guidance on robustness and fairness in particular.

Please contact us at hello@4fivelabs.com with any questions or feedback on these safety guidelines.

2. Safety Challenges for High-Bandwidth, Open-Ended ML Systems

A common definition for safety of non-AI technologies is "Freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment."[1] For the API, we adopt an amended, broader version of this definition:

> Freedom from those conditions that can cause physical, psychological, or social harm to people, including but not limited to death, injury, illness, distress, misinformation, or radicalization, damage to or loss of equipment or property, or damage to the environment.

Our guidance for API safety is informed by considerations particular to systems with machine learning (ML) components that can have high-bandwidth, open-ended interactions with people (e.g. via natural language).

- **ML components have limited robustness.** ML components can only be expected to provide reasonable outputs when given inputs similar to the ones present in the training data. Even if an ML system is considered safe when operated under conditions similar to training data, human operators can provide unfamiliar inputs that put the system into an unsafe state, and it is often not obvious to an operator what inputs will or won't lead to unsafe behavior. Open-ended ML systems that interact with human operators in the

[1] NPR 8715.3C and MIL-STD-882D

general public (e.g. in a question-answering application) are also potentially susceptible to adversarial inputs from malicious operators who deliberately try to put the system into an undesired state.

- **ML components are biased.** ML components reflect the values and biases present in the training data, and systems using ML components—especially systems that interact with people in open-ended ways—can perpetuate or amplify those values. Safety concerns arise when the values embedded into ML systems are harmful to individuals, groups of people, or important institutions. For ML components like the API that are trained on massive amounts of value-laden training data collected from public sources, the scale of the training data and complex social factors make it impossible to completely excise harmful values.
- **Open-ended systems have large surface areas for risk.** Systems that have high-bandwidth interactions with end users, like natural language dialogue or question-answering, can be used for virtually any purpose. This makes it impossible to exhaustively enumerate and mitigate all potential safety risks in advance. Instead, we recommend an approach focused on considering broad categories of potential harm, continuous detection and response for incidents of harm, and continuous integration of new mitigations as needs become apparent.
- **Safety is a moving target for ML systems.** The safety characteristics of ML systems change every time the ML components are updated, for example if they are retrained with new data, or if new components are trained from scratch with novel architectures. Since ML is an area of active research and new levels of performance are routinely unlocked as research advances, ML system designers should count on frequent updates to the ML components and make plans to perform continuous safety analysis.

3. Harms to Consider in Risk Analysis

Below, we give examples of potential harms (or paths to harm) that may arise in systems involving the API as a component. Notably: this list is non-exhaustive, and not every category will apply to every system that uses the API. Additionally, these harms vary in terms of their likelihood, severity, and potential scale. Likelihood, severity, and scale are critical variables when assessing the overall seriousness of a particular safety risk and informing appropriate mitigations.

- **Frustration.** The system generates frustration on the part of a user by giving unhelpful responses or voicing an opinion on a hot-button issue.
- **Perpetuating discriminatory attitudes.** The system persuades users to believe harmful things about groups of people, e.g. by using racist or sexist language.
- **Providing false information.** The system presents false information to users on matters that are safety-critical or health-critical, e.g. giving an incorrect response to a user asking whether they are experiencing a medical emergency and should seek care.

- **Individual distress.** The system creates outputs that are distressing to an individual, e.g. by encouraging self-destructive behavior (like gambling, substance abuse, or self-harm) or damaging their self-esteem.
- **Incitement to violence.** The system persuades users to engage in violent behavior against any other person or group.
- **Physical injury, property damage, or environment damage (in systems with physical actuators).** If a system using the API is connected to physical actuators with the potential to cause harm, the system is safety-critical, and physically damaging failures could result from unanticipated behavior in the API.

4. General Guidance on Risk Mitigation

In this section, we give general guidelines for mitigating potential risks from API usage. We focus on guidance that is most relevant to production settings, i.e. where the scale of impacts of a system (both positive and negative) will be greater than in research/prototyping phases. Sections 5 and 6 below zoom in on safety issues related to robustness and fairness.

- **Test regularly and adversarially.** Develop use-case specific tests for risks you identify as plausible in the context of your application. Make use of both manual and automated tests where applicable. Probe your model in an adversarial fashion--i.e. attempt to create inputs that would cause the model to fail, because if you discover them before your users do you can put up a safeguard before the system goes live.
- **Be transparent about limitations.** Carefully modulate the trust that end users have in outputs from your system (e.g. via disclosing that outputs might be inaccurate or offensive). Clarifying the precise extent of automation involved in a particular application can also help calibrate users on what to expect.[2]
- **Leverage user feedback.** Create scalable mechanisms to get feedback from end users regarding the quality and impact of outputs.
- **Choose use cases carefully.** Avoid use cases which would require a higher degree of reliability than is attainable with language models (e.g. safety-critical applications).
- **Monitor operation.** Maintain awareness of your system's performance characteristics over time, through automated and manual approaches, especially after any changes to the underlying language model or other components of your pipeline.
- **Consider/test the full stack.** If your application has multiple interacting ML components, make sure you test your application on the distribution of inputs that will actually be seen during operation. For example, suppose you have an ML module that translates a request from an end user into a query that goes to the API. If you test the API on a bunch of queries generated directly by end users (and not by the intermediate ML module you use in production), a passing grade for the component on that distribution doesn't mean the composed system would be safe.

[2]For a recently proposed framework for describing levels of automation in language generation, see: https://arxiv.org/abs/2006.06295

- **Consider malicious uses.** Customers should consider ways in which their application might be deliberately misused and take efforts to mitigate any such risks. Exposure to malicious use will vary across use cases. Examples of potential risks include users deliberately triggering offensive text that could be seen by others, users poisoning the system's training data by submitting false user feedback or offensive inputs, and users leveraging a platform to generate spam or disinformation. If significant misuse risks are identified, customers should take steps to mitigate them--for example, by monitoring for anomalous usage patterns or establishing rate limits for end users.

5. Guidance on Robustness

"Robustness" here refers to a system reliably working as intended and expected in a given context. API customers should make reasonable efforts to ensure that their application is as robust as required for safe use and should ensure that such robustness is maintained over time.

Robustness is challenging. Language models such as those included in the API are useful for a range of purposes but can fail in unexpected ways due to limited world knowledge. These failures might be visible, such as generation of irrelevant or obviously incorrect text, or invisible, such as failing to find a relevant result when using API-powered search. The risks associated with using the API will vary substantially across use cases, though some general categories to consider include: generation of text that is irrelevant to the context (providing more context will make this less likely); generation of inaccurate text due to a gap in the API's world knowledge; continuation of a context that is offensive; and inaccurate classification of text.

Context matters a lot. Customers should bear in mind that because the models underlying the API attempt to guess should come next in a sequence of text, API outputs are heavily dependent on the context provided to the model. Providing additional context to the model (such as by giving a few high-quality examples of desired behavior prior to the new input) can make it easier to steer model outputs in desired directions.

Spot robustness issues prior to deployment. Customers should manually evaluate model outputs for each use case being considered, with those outputs being generated over a range of representative inputs as well as some adversarial inputs.

Encourage human oversight. Even with substantial efforts to increase robustness, some failures will likely still occur. As such, API customers should encourage human scrutiny of outputs in order to prevent excessive deference to model outputs.

Keep testing. One way in which the API might not perform as intended, despite initially promising performance, is if the input distribution shifts over time. Additionally, 4Five Labs may provide improved versions of models over time, and customers should ensure that such versions continue to perform well in a given context.

6. Guidance on Fairness

“Fairness” here means ensuring that the API does not either have degraded performance for users based on their demographics or produce text that is prejudiced against certain demographic groups. API customers should take reasonable steps to identify and reduce foreseeable harms associated with demographic biases in the API.

Fairness is very challenging. Due to the API being trained on human data, our models exhibit various biases, including but not limited to biases related to gender, race, and religion. For example: the API is trained largely on text in the English language, and is best suited for classifying, searching, summarizing, or generating such text. The API will by default perform worse on inputs that are different from the data distribution it is trained on, including non-English languages as well as specific dialects of English that are not as well-represented in our training data. 4Five Labs will provide customers a baseline analysis of some biases we have found, though such analysis is not comprehensive; customers should consider fairness issues that may be especially salient in the context of their use case, even if they are not discussed in our baseline analysis. Note that context matters greatly here: providing the API with insufficient context to guide its generations, or providing it with context that relates to sensitive topics, will make offensive outputs more likely.

Characterize fairness risks before deployment. Users should consider their customer base and the range of inputs that they will be using with the API, and should evaluate the performance of the API on a wide range of potential inputs in order to identify cases where the API’s performance might drop.

Block listing and detection tools can help but aren’t panaceas. 4Five Labs will provide a provisional blocklist (list of terms that the API will be blocked from outputting) and a detection tool aimed at flagging potentially sensitive inputs or outputs. These tools are intended to help customers mitigate the risks of offensive outputs. Customers should consider whether their use case calls for the use of such techniques, and if so, how they might be modified to best fit their use case (e.g. trading off false positives and false negatives). Customers should bear in mind that these tools are not a panacea for eliminating all potentially offensive outputs--offensive outputs using other “safe” words may still be generated, and legitimate outputs might be blocked.[3]

---

[3] A recognized challenge with toxicity detection tools is that they can sometimes have the unintended effect of silencing already marginalized voices, further demonstrating the need for use-case-specific assessment in the use of such tools.https://homes.cs.washington.edu/~msap/pdfs/sap2019risk.pdfOne potential way of addressing such tradeoffs (for some applications) is having blocklisting and detection activated as a default but toggleable by users.