CycloneDX

Authoritative

Guide to AI/ML-BOM

Drive Transparency, Compliance, and Security Across the AI Supply Chain

OWASP

First Edition

Table of Contents

About the Guide

CycloneDX is a modern standard for the software supply chain. It has been ratified as ECMA-424 by Ecma International.

The content in this guide results from the work of the CycloneDX AI/ML Working Group with continuous community feedback and input from leading experts in the field. This guide would not be possible without valuable feedback from peers at the CycloneDX Industry Working Group (IWG), the CycloneDX Core Working Group (CWG), the many CycloneDX Feature Working Groups (FWG), Ecma International Technical Committee 54, and a global network of contributors and supporters.

Copyright and License

First Edition, 03 March 2026

Version	Changes	Updated On	Updated By
First Edition	Initial Release	2026-03-03	CycloneDX AI/ML Working Group

Preface

Welcome to the Authoritative Guide series by the OWASP Foundation and OWASP CycloneDX. In this series, we aim to provide comprehensive insights and practical guidance, ensuring that security professionals, developers, and organizations alike have access to the latest best practices and methodologies.

At the heart of the OWASP Foundation lies a commitment to inclusivity and openness. We firmly believe that everyone deserves a seat at the table when shaping the future of cybersecurity standards. Our collaborative model fosters an environment where diverse perspectives converge to drive innovation and excellence.

In line with this ethos, the OWASP Foundation has partnered with Ecma International to create an inclusive, community-driven ecosystem for the development of security standards. This collaboration empowers individuals to contribute their expertise and insights, ensuring that standards like CycloneDX reflect the collective wisdom of the global cybersecurity community.

One standout example of this model is OWASP CycloneDX, which has been ratified as an Ecma International standard and is now known as ECMA-424. By leveraging the strengths of both organizations, CycloneDX serves as a cornerstone of security best practices, providing organizations with a universal standard for software and system transparency.

As you embark on your journey through this Authoritative Guide, we encourage you to engage actively with the content and join us in shaping the future of cybersecurity standards. Together, we can build a safer and more resilient digital world for all.

Andrew van der Stock
Executive Director, OWASP Foundation

The Innovative History of OWASP CycloneDX

OWASP CycloneDX has carved a legacy steeped in innovation, collaboration, and a commitment to openness. OWASP continues to advance software and system transparency standards, prioritizing capabilities that facilitate risk reduction.

October 2025

OWASP CycloneDX v1.7

First specification supporting citations that improve traceability, attribution, and auditability, and comprehensive support for patents and patent families to address intellectual property transparency.

June 2024

International Standardization

CycloneDX v1.6 ratified as an Ecma International standard and published as ECMA-424.

April 2024

OWASP CycloneDX v1.6

First specification to support cryptographic assets for Post-Quantum Cryptography (PQC) readiness and first general-purpose attestation specification to digitally transform audit and attestation workflows.

December 2023

Ecma TC54 Established

First working group chartered with holistic supply chain goals of standardizing core data formats, APIs, and algorithms that advance software and system transparency.

June 2023

OWASP CycloneDX v1.5

First specification to support AI Transparency, configuration and data components, and formulation describing how components were created, tested, trained, evaluated, and deployed.

January 2022

OWASP CycloneDX v1.4

First specification to introduce vulnerability sharing and transparency, including Vulnerability Disclosure Reports (VDR) and Vulnerability Exploitability eXchange (VEX).

May 2021

OWASP CycloneDX v1.3

First specification to incorporate support for composition completeness surpassing NTIA's framing of "known unknowns".

May 2020

OWASP CycloneDX v1.2

First specification to incorporate SWID (ISO/IEC 19770-2:2015) and services into inventory including data classifications, providers, and relationships between services and components.

March 2019

OWASP CycloneDX v1.1

First specification with complete pedigree support describing component lineage and the commits, patches, and diffs which make a forked version unique.

March 2018

OWASP CycloneDX v1.0

First general-purpose, security-focused Bill of Materials standard supporting software and hardware components. Introduced the world to Package-URL for software security use cases.

Source: https://tc54.org/history

Introduction

CycloneDX is a modern standard for the software supply chain. At its core, CycloneDX is a general-purpose Bill-of-Materials (BOM) standard capable of representing software, hardware, services, and other types of inventory.

CycloneDX is notably an OWASP flagship project, has a formal standardization process and governance model through Ecma Technical Committee 54, and is supported by the global information security community.

What is an ML-BOM?

An ML-BOM (Machine Learning Bill of Materials) is a CycloneDX BOM document designed to address the unique complexities and risks of AI/ML systems. It provides a detailed inventory of all components, configurations, and processes involved in the development, training, deployment, and hosting (i.e., via hardware/software stacks and frameworks) of a machine learning model.

The primary purpose of an ML-BOM is to ensure transparency, traceability, security, and compliance throughout an ML model's lifecycle.

Why ML-BOMs are Important

ML-BOMs address critical challenges in the machine learning supply chain:

- Security & Vulnerability Management: Help identify security risks, such as malicious (open-source) models or vulnerable dependencies, before they are integrated into production applications.
- Governance & Compliance: Provide documentation for audits or formal informational requests based upon requirements from emerging global AI regulations such as the European Union's Cyber Resilience Act (EU CRA), including specifics for AI models and systems from the complementary EU AI Act, as well as for voluntary, guidance-focused frameworks such as the NIST AI Risk Management Framework.
- Risk Mitigation: Enable teams to track data lineage, helping to identify and eliminate potential data quality issues, privacy risks, or unwanted biases that could affect the model's performance and fairness.
- Reproducibility & Explainability: Show adherence to software development lifecycle best practices by providing a detailed record of components and training processes such that developers are able to reproduce models (via training from datasets) and their benchmarks in order to validate claims of model accuracy and adherence to ethical considerations.

Core Concepts and Considerations

Key Components of an ML-BOM

An ML-BOM typically documents the identifying elements, architecture, components, and its supply chain along with any configurations and developmental or executional considerations, inclusive of the following areas:

- **Model identifiers**: Identifying information such as the model's Package URL (PURL) (e.g., from Huggingface pkg:huggingface/distilbert-base-uncased@043235d6088ecd3dd5fb5ca3592b6913fd51602) or other domain-specific identifiers within other registries.
- **Model metadata**: Descriptive details such as the model's name, version, license, developer, purpose, use cases, architecture, (hyper)parameters and any additional identifying elements.
- **Model architecture**: Description of the composition of the model's neural network including configurations, layers, input/output parameters, attention mechanisms, etc. used at network processing stages.
- **Datasets**: Description of datasets, as CycloneDX data components, used for training and testing of the associated model. This includes data sources, selection criteria, acquisition methods, preprocessing steps, and more.
- **Tokenizers and prompt templates**: Descriptive details of specific tokenizers (e.g., libraries, files, configurations) and prompt templates used to train and/or interact with the model during runtime.
- **Hardware, software & frameworks**: A list of all hardware and software components including libraries, packages, frameworks (e.g., TensorFlow, PyTorch, Huggingface), along with specific versions and associated licenses used in aspects of the model's lifecycle. This informational category may also include operational and application aspects of models (perhaps as agents) used within compositional frameworks and workflows, along with the protocols used for communication.
- **Training & testing details**: Information about the computational environment and systems (software, hardware, operating system, and GPUs) used for training or evaluation along with necessary configurations, hyperparameters, and evaluation metrics.
- **Intended use & ethical considerations**: Documentation of the model's intended use, known limitations, safety guardrails, and ethical considerations.
- **Environmental impacts**: Documentation of the resource needed to train or execute the model which have an environmental impact or cost (e.g., data center energy and water cooling cost details).

ML-BOM Design and Best Practices

Overview

A Machine Learning Bill-of-Materials (MLBOM or ML-BOM) is an object model to describe a machine learning model, its compositional assets, and other descriptive information often used to assess risk and compliance. Support for MLBOM is included in CycloneDX v1.5 and higher.

An MLBOM relies on many of the common, core elements of the CycloneDX schema, as well as unique aspects specific to ML components, their architectures, metadata, training, and other information used to gauge regulatory compliance.

This guide will provide specifics and best practices for conveying ML-related information using the CycloneDX schema.

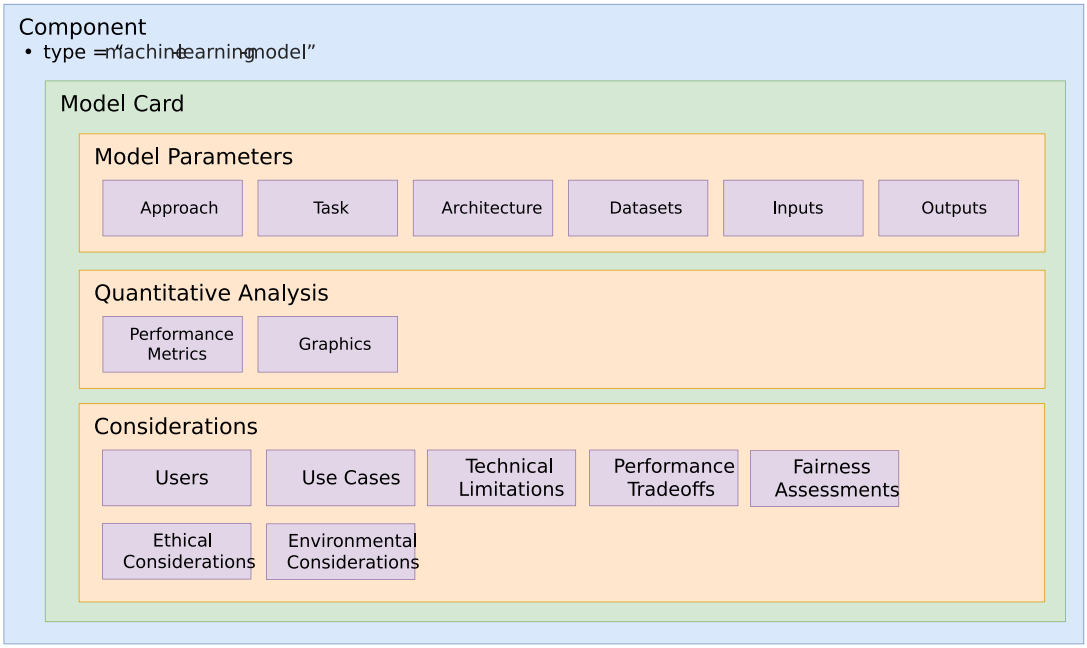
The Core Concepts listed in the previous section will be used to provide details, best practices, and examples for providing the corresponding information using CycloneDX schema objects.

For convenience, here are links to the specific sections for each of those informational areas:

- Anatomy of an ML-BOM
- Declaring ML Models
    - Describing models as components
    - Model repositories as components
    - Model identifiers
    - Describing a model repository as a CycloneDX assembly
    - Declaring a model's pedigree

Anatomy of an ML-BOM

In CycloneDX, a model is considered a component where general best practices for providing information such as component identification, metadata, provenance, pedigree, etc. should be followed as documented in the CycloneDX Authoritative Guide to SBOM.



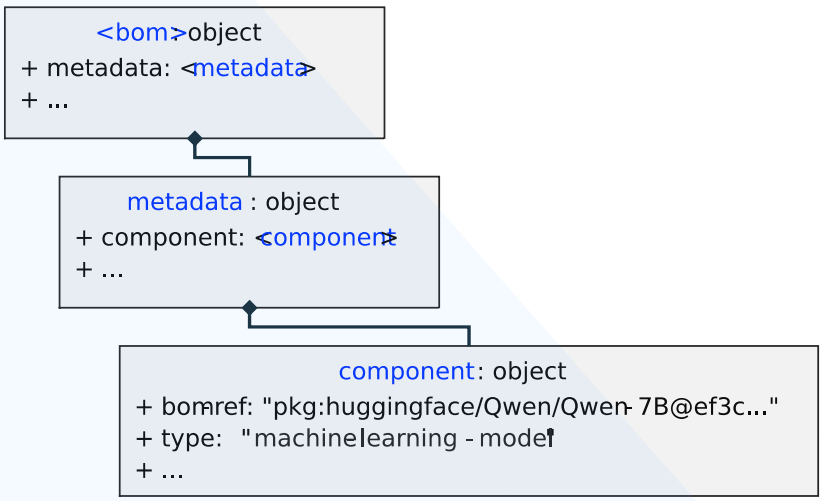
4

Declaring ML models

Describing models as components

A model should always be declared as a CycloneDX component. If the model itself is the subject of the BOM, then the BOM is considered an ML-BOM, and the component representing it would be declared in the top-level BOM metadata object.

The object model's pseudo-schema would look something like this:



Example: Declaring an ML model in an ML-BOM

The CycloneDX JSON pseudocode below shows how an ML model would be declared as the "subject" component of an ML-BOM within the top-level metadata:

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  "bomFormat": "CycloneDX",
  "specVersion": "1.7",
  "serialNumber": "urn:uuid:ec45525e-516c-4405-9de3-4fbdaef7f09a",
  "version": 1,
  "metadata":
  {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
      "purl": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9c57b252f3149c1408daf4d649ec8b6c85",
      "version": "ef3c5c9c57b252f3149c1408daf4d649ec8b6c85",
      // ...
    }
    // ...
  }
  // ...
}
```

Field discussion

- **bom-ref** - Please note the bom-ref value includes the first seven characters of the larger hash value from the purl component identifier which is sufficient for local identification within the BOM itself.

Model repositories as components

When referencing an ML model as a component, it typically means you are referencing a **model repository** comprised of metadata and a set of files (e.g., pre-trained tensor data in various formats, model configurations, tokenizers, tokenizer configurations, prompt templates, Python code, etc.) which would be selectively used with various, compatible AI or ML applications and frameworks.

If possible, these model repositories should be treated as a software "package" in a Software Bill of Materials (SBOM) when declaring them as machine-learning-model type CycloneDX components.

Example: CycloneDX for the Qwen-7B model repository

The following example shows how the Hugging Face Qwen/Qwen-7B model repository would be declared as a CycloneDX component of type machine-learning-model in a CycloneDX ML-BOM as its subject component.

Since the model repository is hosted on Hugging Face, the Huggingface package type may be used Package URL specification to identify the model.

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...
  "metadata":
  {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
      "purl": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9c57b252f3149c1408daf4d649ec8b6c85",
      "group": "Qwen"
      "manufacturer": "Alibaba Cloud",
      "supplier": "Hugging Face",
      "name": "Qwen/Qwen-7B",
      "version": "ef3c5c9c57b252f3149c1408daf4d649ec8b6c85",
      "description": "Qwen-7B is a Transformer-based large language model, which is pretrained on a large volume of data, including web texts, books, codes, etc.",
      "externalReferences": [
        {
          "type": "vcs",
          "url": "https://huggingface.co/"
        },
        {
          "type": "model-card",
          "url": "https://huggingface.co/Qwen/Qwen-7B"
        }
      ],
      "modelCard": {
        // ...
      }
    }
  }
}
```

Field discussion

This section provides best practice guidance on how the component fields were filled out for this example.

- **bom-ref** - Since a PURL is available, it can also be used as the bom-ref.
- **purl** - The Package URL (PURL) follows the Huggingface package type using a commit hash.
- **manufacturer** - The name of the company which built the Qwen model.
- **group** - In this example, we chose to include the optional group field to acknowledge the specific model repository is part of the Qwen family of models.
- **name** - The model name reflects how the model is identified under Hugging Face using the <owner_name>/<repository_name> format.
- **version** - Models are not always versioned in the way software packages are (e.g., using semver format); however, within repositories such as Huggingface, the version is determined by its version control system's *commit hash*, *tag*, or *branch*. In the above example, the model's commit hash matches the purl value.
- **externalReferences** - Used to provide unambiguous links to component's model repository and originating model card.

- **vcs** - Provides a link to the version control system (i.e., the model provider aka. supplier). In this example, Hugging Face is used to affirm the associated PURL identifier.

- **model-card** - Provides a link to the model's Hugging Face model card which is comprised of mostly unstructured information in the form of a markdown file (i.e., README.md).*The CycloneDX representation of model card information will be detailed in a subsequent section.*

Model identifier(s)

As you can see in the above example, the component has a bom-ref that is also a valid Package URL (PURL) for a "Qwen-7B" model hosted in a Huggingface model repository using the Hugging Face PURL type. When a valid purl value is available for a model, it is recommended that it also be used as its component's bom-ref.

If the model being described by an ML-BOM is instead hosted in a GitHub repository, it can also be referenced using a GitHub Package URL. For example, the ONNX vision model: tiny-yolov2 would have a github PURL type.

Example: JSON for model component with GitHub PURL

Note: The derivative bom-ref, based upon the PURL, is also shown.

```
"component":
{
  "type": "machine-learning-model",
  "purl":
"pkg:github/onnx/models@4c46cd00fbdb7cd30b6c1c17ab54f2e1f4f7b177#validated/vision/object_detection_segmentation/tiny-yolov2/model",
  "bom-ref": "pkg:github/onnx/models@244fd47#tiny-yolov2/model"
  // ...
}
```

Adding domain-specific identifiers

Organizations that produce BOMs for hardware or software components they produce may have multiple domain-specific identifiers for the same component. In these cases, it is best practice to register (reserve) an official namespace for these domains with the **Error! Hyperlink reference not valid.**, which is the authoritative source of official namespaces used in CycloneDX properties.

Example:

The following example shows how a registered name for a fictional company, ACME, which registered the namespace acme, could provide a property to identify one of its internal ML models.

```
"component": {
  "properties": [
    {
      "name": "acme:research:model:llm:id",
      "value": "MODEL-ID-12345-INTERNAL"
    },
    // ...
  ],
  // ...
}
```

Identifying a specific model quantization

Some model repositories may contain different quantizations to choose from, which optimize for different target inference runtimes and hardware footprints.

In general, these are referenceable as (often single) files within a model repository, each of which can be described as a CycloneDX component, as shown in the next section Describing a model repository as a CycloneDX assembly.

Example: Qwen/Qwen3-8B-GGUF

This example uses the model repository Qwen/Qwen3-8B-GGUF, which contains several quantizations of the Qwen3-8B model (published originally in a non-quantized format elsewhere) in GGUF format.

These quantized GGUF models are each individual files in the repository:

- Qwen3-8B-Q4_K_M.gguf
- Qwen3-8B-Q5_0.gguf
- Qwen3-8B-Q6_K.gguf
- Qwen3-8B-Q8_0.gguf

Each can be specifically identified in a CycloneDX component using a Package URL (PURL). For example, the Qwen3-8B-Q4_K_M.gguf model would be declared as follows:

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  "bomFormat": "CycloneDX",
  "specVersion": "1.7",
  "serialNumber": "urn:uuid:1ad676cb-6b40-4068-ae91-ebd1533dbf58",
  "version": 1,
  // ...,
  "components": [
    {
      "name": "Qwen3-8B-Q4_K_M.gguf",
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/Qwen/Qwen3-8B-GGUF@7c41481#Qwen3-8B-Q4_K_M.gguf",
      "purl": "pkg:huggingface/Qwen/Qwen3-8B-
GGUF@7c41481f57cb95916b40956ab2f0b139b296d974#Qwen3-8B-Q4_K_M.gguf",
      "version": "7c41481f57cb95916b40956ab2f0b139b296d974",
      // ...
    }
  ],
  // ...
}
```

Field discussion

- **type** - the type has the value machine-learning-model since the single file contains all the information (e.g., default configuration parameters, references to architectures and tokenizers, prompt template, etc.) needed to run the model in GGUF inference frameworks.

Describing a model repository as a CycloneDX assembly

CycloneDX allows for declarations of software compositions (e.g., hardware products, software applications, packages, libraries, archives, etc.).

In the case of a model repository like those hosted in Hugging Face, one can describe the files that comprise it as a composition with an ML-BOM. Specifically, it would be declared as a composition of an assembly type.

Specifically, a component entry would be created for each file and declared in the ML-BOM's components array hierarchically under the model's component then declare the assembly relationship

within within the BOM's compositions array under assemblies by providing the bom-ref link to the model component that contains the hierarchy of the constituting (file) components within the model repository.

Example: Qwen/Qwen-7B model repository files

If we look inside the repository for the Qwen/Qwen-7B model in Huggingface, we see the complete list of files that make up the "model" in its repository:

CycloneDX for the Qwen/Qwen-7B assembly

The simplified JSON below shows how to declare a few files from the model repository's complete file list within the BOM's component under the model's metadata declaration.

> *Note: In the JSON below, we use the Package URL (PURL) syntax to provide the additional path (with the model repository or "package") to each individual file by appending it using the # hash symbol as a separator. Also, notice that the commit hash (identifier) varies per file.*

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
   // ...,
   "metadata":
   {
     "component":
     {
       "type": "machine-learning-model",
       "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
       // ...
       "components": [
         {
             "type": "file",
             "name": "config.json",
             "description": "Model configuration file using the 'QWenLMHeadModel' model class in Hugging Face Transformers",
             "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@e7a368b#config.json",
             "purl": "pkg:huggingface/Qwen/Qwen-7B@e7a368b077437 0edec29674e7c51f52fc7663f59#config.json",
             // ...
         },
         {
             "type": "file",
             "name": "configuration_qwen.py",
             "description": "Python 'QWenConfig' class implementation for the Qwen-7B model using Hugging Face Transformers",
             "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@a6ca629#configuration_qwen.py",
             "purl": "pkg:huggingface/Qwen/Qwen-7B@a6ca629d063f56f34d184852301e8852a7afbd58#configuration_qwen.py",
             // ...
         },
         {
             "type": "data",
             "name": "model-00001-of-00008.safetensors",
             "description": "Model tensor data (01 of 08)",
             "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@abcb6d6#model-00001-of-00008.safetensors",
             "purl": "pkg:huggingface/Qwen/Qwen-7B@abcb6d6d8ec63ce606f816e2d08072da6309f965#model-00001-of-00008.safetensors",
             "data": {
               "type": "dataset",
               // ...
             }
```

```
            // ...
          },
          {
            "type": "data",
            "name": "model-00002-of-00008.safetensors",
            "description": "Model tensor data (02 of 08)",
            "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@abcb6#model-00002-of-00008.safetensors",
            "purl": "pkg:huggingface/Qwen/Qwen-7B@abcb6d6d8ec63ce606f816e2d08072da6309f965#model-
00002-of-00008.safetensors",
            "data": {
              "type": "dataset",
              // ...
            }
            // ...
          },
          // ...
        ]
      }
    }
  // ...
}
```

Then the model component's new hierarchy of composing files would be described as an assembly composition as follows:

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...,
  "composition": [
    {
      "aggregate": "complete",
      "assemblies": [
        "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
      ]
    }
  ],
  // ...
}
```

Discussion of composition fields

- **aggregate** - Note the composition aggregate value is assigned to be "complete" since all constituent files are known and declared in the ML-BOM as part of the model component's components hierarchy.

Declaring a model's pedigree

ML models are often derived from existing, pre-trained models to optimize performance, reduce resource consumption, and adapt to specialized tasks without training from scratch. Some reasons for this include:

- **Fine-Tuning**: Specialized adaptation where a general model (e.g., LLM) is retrained on a smaller, targeted dataset to improve performance for specific domains.
- **Quantization**: Reduces model size and increases inference speed by mapping parameters to lower-precision tensor formats (e.g., from FP32 to int8 or Q4_K_M precision), which also lowers energy consumption for edge devices.

- **Format Conversions**: Transforming models between frameworks (e.g., PyTorch to ONNX) ensures interoperability, allowing deployment on different frameworks and accelerators.
- **Pruning**: Derives a smaller model by removing redundant or less important parameters (weights) that do not significantly contribute to output accuracy.
- **Adapters**: Adding small, trainable layers (adapters) to a frozen base model to adapt it to new tasks without changing the original, large model weights, saving on storage for multi-task scenarios.

It is important to capture any of these transformations in the model's lineage ("pedigree") or in an ML-BOM. This should be accomplished via the CycloneDX pedigree object and describing a model's ancestors as a hierarchical graph.

Example: Declaring the finetuning of llama3 model for a coding variant

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...,
  "metadata": {
    "component": {
      "type": "machine-learning-model",
      "name": "unsloth/Llama-3.2-3B-Instruct",
      "purl": "pkg:huggingface/unsloth/Llama-3.2-3B-Instruct@1.0.0",
      "bom-ref": "pkg:huggingface/unsloth/Llama-3.2-3B-Instruct@1.0.0",
      "publisher": "Unsloth",
      "description": "A pre-optimized, specialized versions of the meta-llama/Llama-3.2-3B-Instruct model designed to work seamlessly with Unsloth's training framework",
      // ...,
      "pedigree": {
        "ancestors": [
          {
            "type": "machine-learning-model",
            "name": "meta-llama/Llama-3.2-3B-Instruct",
            "publisher": "Meta",
            "purl": "pkg:huggingface/meta-llama/Llama-3.2-3B-Instruct",
            "description": "The original base model from Meta Llama used for fine-tuning."
          }
        ]
      }
    }
  }
}
```

Field discussion

- **ancestors** - ancestors entries are themselves CycloneDX component objects. It should be noted that these models may have their own ML-BOMs, which can be located via their identifiers (e.g., purl) or via externalReferences for readers to follow.

Declaring known descendents

If, at the time an ML-BOM is created for a model, its downstream model variants (e.g., finetunings, quantizations, etc., derived from the model) are known, these can also be recorded within the pedigree object as descendants in a similar manner.

Model cards

A model card describes the intended uses of a machine learning model and potential limitations, including biases and ethical considerations. Model cards typically include the training parameters, the datasets used to train the model, performance metrics, and other relevant data useful for ML transparency. This object *SHOULD* be specified for any component of type machine-learning-model and must not be specified for other component types.

Throughout the model card sections of this guide, we will show how to use the existing schema to encode information from model cards in a more current and robust way.

Overview

This section describes the design and best practices when providing information for a CycloneDX modelCard in an ML-BOM as part of the model's CycloneDX component definition.

For convenience, here are links to the specific sections for each of those informational areas:

- Model parameters
  - Model metadata
    - Approach
    - Task
    - Architecture family
    - Model architecture
  - Datasets
  - Inputs & Outputs
  - Declaring other properties
    - Configuration parameters & hyperparameters
- Quantitative analysis
  - Benchmarks
  - Metrics
    - Performance metrics
  - Graphics
- Considerations
  - Users & use cases
  - Technical limitations
  - Performance tradeoffs
  - Fairness assessments

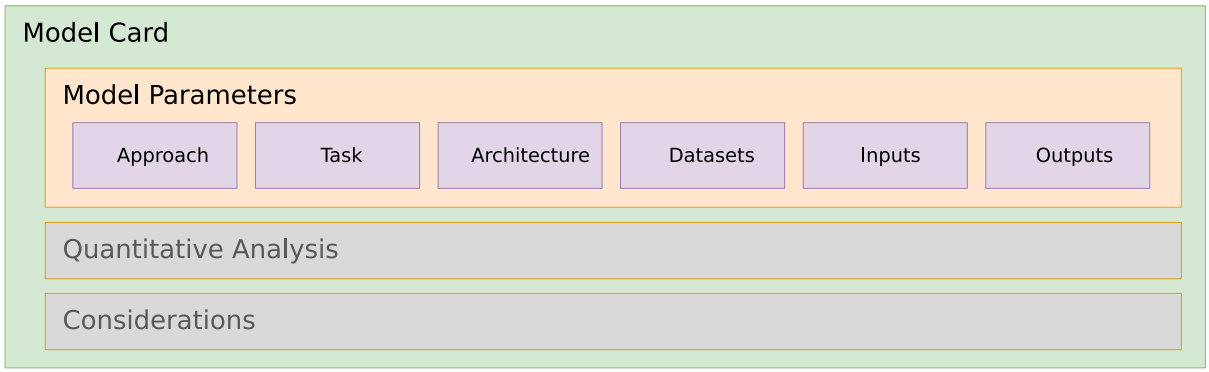
- Ethical considerations
- Environmental impact consideration
  - Energy consumptions
- Additional model-related information
  - Using CycloneDX AI/ML properties
  - Tokenizers and prompt templates
  - Including manufacturing information for the ML model

Design notes

Please note that at the time of initial development, the CycloneDX model card schema was heavily influenced by Tensorflow ModelCard Toolkit and specifically its ModelCard fields since it was one of the few frameworks available for reference.

Since that time, the AI/ML landscape has progressed rapidly, both in the design of model architectures and in the increased emphasis on providing model disclosure to comply with governmental regulations. *In order to account for these changes, the CycloneDX community plans to provide significant improvements and normative guidance in future versions of the specification.*

Model parameters



This section will feature guidance on filling out information in the Cyclone model card's modelParameters object and its subcomponents, including:

- Model metadata
    - Approach - The overall approach to learning used by the model for problem solving.
    - Task - Directly influences the input and/or output. Examples include classification, regression, clustering, etc.
    - Architecture family - The model architecture family such as a Transformer network, Convolutional Neural Network (CNN), residual neural network (RNN), LSTM neural network, etc.
    - Model architecture - The specific architecture of the model such as Transformer, GPT-1, ResNet-50, YOLOv3, etc.
    - External references
- Datasets - The datasets used to train and evaluate the model.
    - Declaring datasets
- Inputs & Outputs - Describes the input and output data types (formats) of the model.
- Declaring other properties
    - Configuration parameters & hyperparameters

Model metadata

The modelCard fields, grouped in this section, are intended to describe some of the metadata used to classify the associated ML model.

Approach

Describes the general learning approach used to train the model. Currently, the approach is simply described by a single type field, which has the following supported values:

Type	Description
supervised	Supervised machine learning involves training an algorithm on labeled data to predict or classify new data based on the patterns learned from the labeled examples.
unsupervised	Unsupervised machine learning involves training algorithms on unlabeled data to discover patterns, structures, or relationships without explicit guidance, allowing the model to identify inherent structures or clusters within the data.
reinforcement-learning	Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards, through trial and error.
semi-supervised	Semi-supervised machine learning utilizes a combination of labeled and unlabeled data during training to improve model performance, leveraging the benefits of both supervised and unsupervised learning techniques.
self-supervised	Self-supervised machine learning involves training models to predict parts of the input data from other parts of the same data, without requiring external labels, enabling learning from large amounts of unlabeled data.

Please note that links to external documentation detailing the model training approach (and other information) can be provided using external references, which are discussed later in this section.

Task

Describes the primary task (or goal) of the machine learning model. Some examples include:

- Anomaly Detection: Identifying outliers or unusual patterns in data.
- Classification: Categorizing inputs into predefined labels (e.g., spam/not-spam, image recognition).
- Clustering: Grouping unlabeled data based on similar characteristics (e.g., customer segmentation).
- Dimensionality Reduction: Simplifying complex data by reducing the number of variables while preserving core information.
- Generation: Creating new data based upon prompted instructions (e.g., Large Language Models (LLMs), image or audio diffusion models, etc.).
- Recommendation/Association: Finding relationships between items (e.g., "users who bought this also bought...").
- Regression: Predicting continuous numerical values (e.g., house prices, temperature forecasting).

Architecture family

An architecture family defines the model's neural network's structural and data-processing methodology. It typically does not describe a single model but rather the general design of the neural network (NN), the mathematical approach, context, attention mechanisms, and the like. It should provide insight to those versed in the field on how the model is constructed.

The model architecture family field should include descriptive names of neural network architectures recognizable to those in the field of Machine Learning (ML).

Some examples of commonly referenced neural network (NN) architecture families include:

- Transformers - an architecture designed to process sequential data (like text, speech, or images) in parallel rather than in order.
- Convolutional Neural Network (CNN) - an architecture designed to process efficiently detect patterns (like edges, shapes, and textures) to typically classify or analyze visual (videos/image) or auditory data, but can applied to text analysis, behavioral patterns and more.
- Recurrent Neural Network (RNN) - an architecture designed for processing sequential data like text, speech, and time series, typically used for tasks where order matters, such as language translation, speech recognition and time-series forecasting.
- Long Short-Term Memory (LSTM) - a specialized variant of a Recurrent Neural Network (RNN) architecture designed specifically to overcome the limitations of traditional RNNs in learning long-term dependencies.
- Gated Recurrent Units (GRUs) a specialized variant of a Recurrent Neural Network (RNN) architecture designed to overcome challenges like the vanishing gradient problem and enhance the modeling of long-term dependencies in sequential datasets.
- Generative Adversarial Networks (GANs) - an architecture used to train two neural networks, a *Generator* and a *Discriminator*, to compete against each other to generate more authentic data from a starting training dataset. The Generator tries to fool the Discriminator by creating fake data, while the Discriminator tries to identify fakes, leading to continuous improvement in data quality.

Again, the list above represents architecture families commonly referenced in research to establish an understanding of general model design; however, the architectural landscape continues to grow as researchers specialize and optimize for different use cases, goals, and datasets.

Model architecture

The model architecture field is intended to include specific keywords to identify an implementation (class or library) or technical descriptors of the architectural "blueprint" needed to run a specific model.

These are typically found in one of several locations relative to the model:

- Model Card: the associated "model card" (e.g., README.md in Hugging Face) may contain a mentions of specific class names like LlamaForCausalLM, BertModel, or VisionTransformer.
- Framework-Specific Implementation Keywords or tags: Depending on your code environment (PyTorch, TensorFlow, llama.cpp, etc.) that identify specific model code within the platform or environment.
- Framework-Specific Configuration files (e.g., Hugging Face transformer's config.json file): may contain the name of the class or function used to configure the framework for the specific implementation recommended for the associated model.
- Academic Research Papers (e.g., arXiv): may include detailed descriptors of processing algorithms, supported training or inference engines or specific, named implementations

Example: Model card metadata for the Qwen-7B model

This example demonstrates best practices for the Qwen-7B model using information published within and for the model's repository on Hugging Face.

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...,
  "metadata":
  {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
      // ...,
      "modelCard": {
        "modelParameters": {
          "task": "text-generation",
          "architectureFamily": "transformer",
          "modelArchitecture": "QWenLMHeadModel",
          "approach": {
            "type": "supervised"
          },
        // ...
      }
    }
  }
}
```

Field discussion

- modelArchitecture - the value QWenLMHeadModel was located in the model's config.json model configuration file.

Providing links to papers & articles

Most models are fully described in terms of research papers, articles, and other reference documents. In those cases, they should be provided as externalReferences under the component.

Example: "Qwen Technical Report"

This shows how the Qwen research team disclosed comprehensive details about the Qwen model's design, training, implementation, and evaluation as a formal research paper in Cornell University's arXiv scholarly article distribution service.

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  "metadata":
  {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
      // ...,
      "externalReferences": [
        {
          "type": "documentation",
          "url": "https://arxiv.org/abs/2309.16609",
          "comment": "Qwen Technical Report"
        }
      ]
    }
  }
}
```

Datasets

Details the datasets used to train and evaluate the model.

Declaring datasets

Using CycloneDX, there are two methods for providing information about the datasets used to train, test, and evaluate machine learning models.

Specifically, the component modelCard object includes modelParameters, which includes an array of datasets objects, which can be of the following types:

1. In-line information: provides in-line objects that provide for direct description of datasets and some of their typically cited attributes and characteristics.

2. Data component references: provides for the complete description of each dataset as its own CycloneDX component and referenced via its bom-ref.

The next sections will discuss the considerations for each and provide examples of how to use both methods.

Datasets as in-line information

This method simplifies the association between training datasets and model cards, specifically addressing scenarios where data is difficult to reference as an independent component.

Key applications:

- Filtered Data: Documenting specific slices or individual snippets of data used for fine-tuning or testing.

- Private Repositories: Providing transparency via BOMs for non-public datasets in public model cards (e.g, private data used for models in the healthcare or financial services industries).

- Unstructured Sources: Referencing data not housed in traditional databases or management tools (e.g., data within S3 buckets, event data within Security information and event management (SIEM) systems).

Example: Custom health model with private dataset

This example shows a model fine-tuned (by a fictional "ACME Health" company) from the public m42-health/Llama3-Med42-8B model using a private dataset.

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  "bomFormat": "CycloneDX",
  "specVersion": "1.7",
  "serialNumber": "urn:uuid:3e671687-395b-41f5-a30f-a58921a69b79",
  "version": 1,
  "metadata": {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/acme-health/custom-Llama3-Med42-8B@2ee9dc9",
      "purl": "pkg:huggingface/acme-health/custom-Llama3-Med42-8B@2ee9dc99-cc50-4490-9d6e-
9ebf6e39f82f",
      "description": "Customized Med42-v2 large language models (LLMs) which uses the Llama3 architecture and
fine-tuned using private clinical dataset."
      // ...,
      "modelCard": {
        "modelParameters": {
          // ...,
          "datasets": [
            {
              "type": "dataset",
              "name": "UltraFeed-
back dataset",
              "classification": "public",
              "contents": {
                "url": "https://huggingface.co/datasets/openbmb/UltraFeedback"
              }
            },
            //...,
            {
              "type": "dataset",
              "name": "ACME Midwest health data",
              "classification": "private",
              "contents": {
                "url": "https://acme.ai/adatasets/health/patient?region=midwest"
              }
            }
          ],
          // ...
        }
      }
    }
  }
}
```

Field discussion

- **url** - Please note that URLs may be to either public or private resources. For example, in the case of the ACME private dataset above, the URL is likely behind an Access Control Point (ACP) which regulates traffic to the private resource in accordance with the ACME company's governance policies.

Datasets as data component references

This method is preferable for most security and compliance contexts, as it allows for the full expression of provenance, pedigree, attestations, and other contextual information as a full CycloneDX component.

Example: health model with private dataset

This example shows the recommended best practice of declaring the datasets for the base model used in the previous "in-line" example (i.e., m42-health/Llama3-Med42-8B) as their own CycloneDX components.

The public datasets, as documented in the model's research paper, include:

- openbmb/UltraFeedback
- snorkelai/Snorkel-Mistral-PairRM-DPO


```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  "bomFormat": "CycloneDX",
  "specVersion": "1.7",
  "serialNumber": "urn:uuid:eb033070-85d1-45f4-9eb7-f50510f83853",
  "version": 1,
  "metadata": {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/acme-health/custom-Llama3-Med42-8B@ceab7e7",
      "purl": "pkg:huggingface/acme-health/Llama3-Med42-
8B@ceab7e7ee4b9dbde7ba82867f34274db51487d83",
      "description": "an open, clinical large language models (LLM) instruct and preference-tuned by M42 to expand access to medical knowledge. Built off LLaMA-3 and designed to provide high-quality answers to medical questions."
      // ...,
      "modelCard": {
        "modelParameters": {
          // ...,
          "datasets": [
            {
              "ref": "pkg:huggingface/openbmb/UltraFeedback@40b4365"
            },
            {
              "ref": "pkg:huggingface/snorkelai/Snorkel-Mistral-PairRM-DPO@07af5d0a"
            }
          ]
        }
      }
    }
  },
  "components": [
    {
      "name": "UltraFeed-back dataset",
      "type": "data",
      "bom-ref": "pkg:huggingface/openbmb/UltraFeedback@40b4365",
      "purl": "pkg:huggingface/openbmb/UltraFeedback@40b436560ca83a8dba36114c22ab3c66e43f6d5e",
      // ...
    },
    {
      "name": "UltraFeed-back dataset",
      "type": "data",
      "bom-ref": "pkg:huggingface/snorkelai/Snorkel-Mistral-PairRM-DPO@07af5d0a",
      "purl": "pkg:huggingface/snorkelai/Snorkel-Mistral-PairRM-
```

```
DPO@07af5d0a875b4c692dfaff6c675b10af07b45511",
      // ...
    }
  ]
}
```

Inputs & outputs

Describes the input and output data types (formats) of the model.

> ***Note**: The current object used to describe model inputs and outputs is limited to describing the data types strictly used for training and inference. Future revisions of CycloneDX plan to expand these objects to provide more detailed information, especially regarding names, formats, and defaults for model configuration parameters and hyperparameters.*

In order to provide information on model parameters and hyperparameters using the existing CycloneDX schema, it is recommended to follow the best practice as shown in the next section "Declaring other properties" and its "Example: Model parameters & hyperparameters for the Qwen-7B model".

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...,
  "metadata":
  {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
      // ...,
      "modelCard": {
          // ...,
          "inputs": [
            {"format": "string"}
          ],
          "outputs": [
            {"format": "string"}
          ]
      }
    }
  }
}
```

Declaring other properties

Configuration parameters & hyperparameters

In general, model configuration parameters describe values used to configure model processing applications, frameworks, and implementations of model architectures. For example, most models on Hugging Face typically include configuration files for both the models and the tokenizers they are designed to work with, using the Hugging Face Transformers library and its underlying PyTorch framework.

> ***Note**: The CycloneDX ModelParameters were initially based upon Tensorflow ModelCard Toolkit (now archived) which defines ModelParameters that include key-value maps for both inputs and outputs alongside an array of their types; these maps will be accomplished using CycloneDX properties and the CycloneDX Property Taxonomy reserved namespace cdx:ai-ml:model:parameter and cdx:ai-ml:model:hyperparameter as needed.*

Example: Model parameters & hyperparameters for the Qwen-7B model

As shown in the Qwen/Qwen-7B model repository files example in the previous section, we see the model includes several configuration files, including:

- config.json - which contains configuration parameters (as key-value pairs) used for initializing the model's implementation.
- generation_config.json - which contains model hyperparameters (as key-value pairs) and their suggested (default) values used for configuring the model for token generation (inference).

The JSON below shows how a few of the Qwen/Qwen-7B model's parameters, as contained in the config.json configuration file, would be declared within the CycloneDX modelCard object's properties array using the CycloneDX reserved namespace for AI/ML.

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...,
  "metadata":
  {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
      // ...,
      "modelCard": {
        "modelParameters": {
          // ...,
          "properties": [
            {
              "name": "cdx:ai-ml:model:parameter:count",
              "value": "7B"
            },
            {
              "name": "cdx:ai-ml:model:parameter:tune_method",
              "value": "sft"
            },
            {
              "name": "cdx:ai-ml:model:parameter:tune_method",
              "value": "rlhf"
            },
```

```
          {
            "name": "cdx:ai-ml:model:hyperparameter:num_hidden_layers",
            "value": "32"
          },
          {
            "name": "cdx:ai-ml:model:hyperparameter:hidden_size",
            "value": "4096"
          },
          {
            "name": "cdx:ai-ml:model:hyperparameter:context_length",
            "value": "8192"
          },
          {
            "name": "cdx:ai-ml:model:hyperparameter:vocab_size",
            "value": "151936"
          },
          {
            "name": "cdx:ai-ml:model:hyperparameter:quantization",
            "value": "BF16"
          },
          // ...
        ]
      },
      // ...
    }
  }
}
```

Field discussion

Please note that the example above includes only a small set of parameters and hyperparameters that extend the cdx:ai-ml:model:parameter and cdx:ai-ml:model:hyperparameter paths. Actual models may have a more comprehensive set of properties declared.

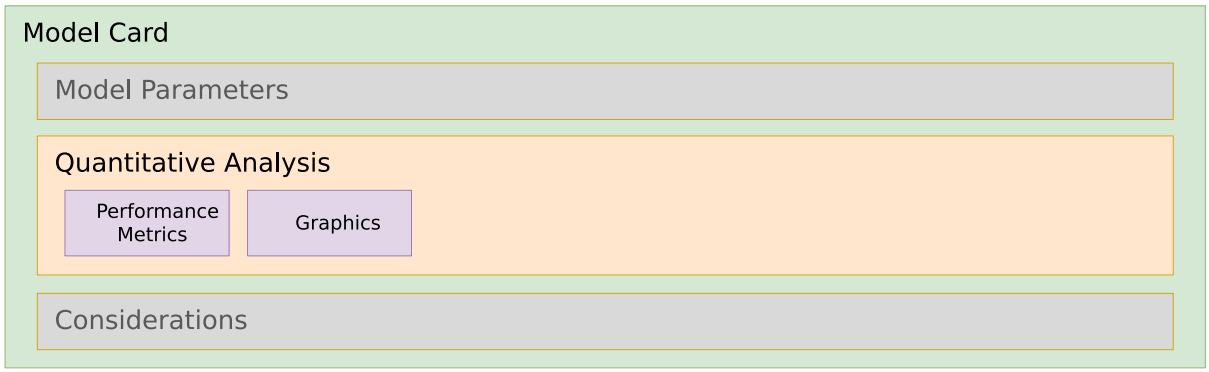
The example model card above contains the following cdx:ai-ml:model:parameter and cdx:ai-ml:model:hyperparameter properties, which are explained below:

- properties
    - context_length - The maximum sequence length the model supports during training and inference.
    - count - Total number of learned parameters in the model.
    - num_hidden_layers - The total number of intermediate (hidden) processing layers situated between the input layer and the output layer of the model.
    - hidden_size - The dimension of the input and output representations (i.e., of the token embeddings) used by the internal (hidden) layers of a model's neural network.
    - tune_methods - Indicates the fine-tuning methods used to develop the model. In this case, sft (Supervised Fine-Tuning) and rlhf (Reinforcement Learning from Human Feedback).
    - quantization - The quantization used for tensor weights which affects model memory usage.
    - vocab_size - The size of the model's vocabulary.

Tokenizer parameters and hyperparameters

The same methodology used to provide model hyperparameter names and values for models can also be applied to model tokenizers by instead using the cdx:ai-ml:tokenizer:hyperparameter path and extending it with a path with a parameter name.

Model quantitative analysis



This section will feature guidance on filling out information in the Cyclone model card's quantitativeAnalysis object and its subcomponents, including:

- Metrics
    - Performance metrics
- Graphics

What is quantitative analysis

Quantitative analysis is the process of using metrics on benchmarks to determine if a model is reliable, safe, or better than another. It involves comparing the metric results against the benchmark standard to assess performance, identify limitations, and track progress over time.

The value of quantitative analysis

- **Numerical Metrics**: Provides measurable data (e.g., error rates, latency, performance scores) rather than subjective feedback.
- **Objective Evaluation**: Provides reproducible, scalable results that can be compared across different models or versions.
- **Pattern & Trend Detection**: Identifies numerical patterns, correlations, and trends within large or complex datasets that might be missed manually.
- **Testing Hypotheses**: Enables the statistical testing of assumptions about model behavior allowing for comparisons against similar models for given tasks.

Benchmarks

Benchmarks are standardized test datasets, scenarios, or tasks that define the "playing field". They provide a consistent environment for evaluating different models and enable the comparison of their metrics across similar models.

Types of machine learning benchmarks

Benchmarks use standardized datasets to objectively compare model quality, efficiency, fairness, and speed, providing a shared baseline for identifying areas for improvement in various categories.

- Large Language Models (LLM) and Natural Language Processing (NLP) (e.g., speech recognition or text classification): These benchmarks evaluate reasoning, knowledge, and generation capabilities. A few examples of datasets used to benchmark these models against different tasks include:
    - General Tasks
        - MMLU, MMLU-Pro (Massive Multitask Language Understanding): Tests knowledge across STEM, humanities, and social sciences.
        - HellaSwag / WinoGrande: Common sense reasoning and pronoun resolution tasks.
        - GLUE: benchmarking resources for training, evaluating, and analyzing natural language understanding systems. GLUE's dataset is available in Hugging Face Hub (nyu-mll/glue) and supports multiple tasks that can be evaluated independently, for example:
            - *ax* - evaluates sentence understanding through Natural Language Inference (NLI) problems.
            - *cola* - The Corpus of Linguistic Acceptability consists of English acceptability judgments drawn from books and journal articles on linguistic theory.
            - *mnli* - The Multi-Genre Natural Language Inference Corpus consists of sentence pairs with textual entailment annotations. Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis.
    - Math/STEM tasks
        - GSM8K (OpenAII, Grade School Math 8K): a dataset of 8.5K high quality linguistically diverse grade school math word problems.
        - MATH-500 (Hugging Face): Benchmarks specifically designed to evaluate mathematical reasoning.
    - Coding Tasks
        - HumanEval (OpenAI): used to evaluate the functional correctness of code generated LLMs. It consists of hand-crafted programming problems designed to test reasoning and code synthesis abilities.
        - MBPP (Mostly Basic Python Problems): Benchmarks for evaluating code generation and programming capabilities.
        - CodeXGLUE (MicroSoft, Code Refinement): Used to evaluate a model's ability to remove (i.e., "fix") bugs from Java code (i.e., refine the code) with accuracy being reported as BLEU scores.
    - Other Tasks

    - IMDB: a large dataset of 50K, highly polarized, movie reviews for NLP sentiment analysis and classification.

- ImageNet: large-scale dataset for computer vision, featuring over 14 million annotated, high-resolution images across thousands of object categories organized by the WordNet hierarchy.
- MathVista: Used to evaluating math reasoning in Visual Contexts. It consists of three datasets, *IQTest*, *FunctionQA*, and *PaperQA*, which are tailored to evaluate visual reasoning on puzzle test figures, algebraic reasoning over functional plots, and scientific reasoning with academic paper figures, respectively.
- MNIST (Modified National Institute of Standards and Technology database): a large database of handwritten digits (glyphs) that is commonly used for training various image processing systems.

Again, the list above contains only a small number of benchmarking datasets that can be used to train and evaluate models.

Metrics

AI benchmarking metrics are standardized, quantitative measures used to evaluate and compare the performance, accuracy, efficiency, and reliability of artificial intelligence models against established, uniform tasks and datasets. They serve as a gauge of progress in capabilities such as reasoning, coding, and language understanding, enabling simple comparisons with similar models.

> ***Note**: Currently, CycloneDX supports declaring metrics relative to performance benchmarks which is the most consistently documented metrics described within producer published model cards.*

Performance metrics

Performance metrics are specific, quantitative measures used to evaluate a model's behavior, such as accuracy, precision, recall, perplexity, or inference speed. They provide the raw, numerical data for analysis.

Common Performance Metrics

- **Accuracy**: Overall correctness; typically represented as a percentage of correct responses to the full set of problems posed by a benchmark's dataset.
- **Precision**: Of predicted positives, how many are correct.
- **Recall** (Sensitivity): Of actual positives, how many are found.
- **F1 Score**: Harmonic mean of *Precision* and *Recall*.

Example: Declaring the MMLU accuracy score for Qwen-7B

The Qwen accuracy scores for various benchmarks are published in the QwenLM/Qwen GitHub repository's README.

This appears as a table that includes all Qwen models, along with other similar models for comparison. Here is the table row for all Qwen-7B benchmarks:

Model	MMLU	C-Eval	GSM8K	MATH	HumanEval	MBPP	BBH	CMMLU
Qwen-7B	58.2	63.5	51.7	11.6	29.9	31.6	45.0	62.2

The MMLU score from the table would be declared as a performance metric as follows:

```
"component":
{
  "type": "machine-learning-model",
  "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
  // ...,
  "modelCard": {
    // ...,
    "quantitativeAnalysis": {
    "performanceMetrics": [
     {
         "type": "MMLU (5-shot)",
         "value": "58.2",
         "confidenceInterval": {
           "lowerBound": "94.28",
           "upperBound": "95.72"
         }
       }
     ]
    }
  }
}
```

Field discussion

- slice - the slice property was omitted indicating the full dataset was used for performance benchmarking.
- confidenceInterval - the values provided reflect Statistical Confidence Interval (Accuracy) for the full MMLU test set (approx. 14,000–15,900 questions) which is 95% ±0.72.

Example: Declaring a GLUE F1 Score

This example shows how to provide an F1 score (i.e., the harmonic mean of precision and recall measurements) for a model's performance on classification tasks within the GLUE benchmark.

```
"quantitativeAnalysis": {
  "performanceMetrics": [
    {
      "type": "GLUE (F1 Score)",
      "value": "0.87",
      "slice": "cola"
    }
  ]
}
```

Field discussion

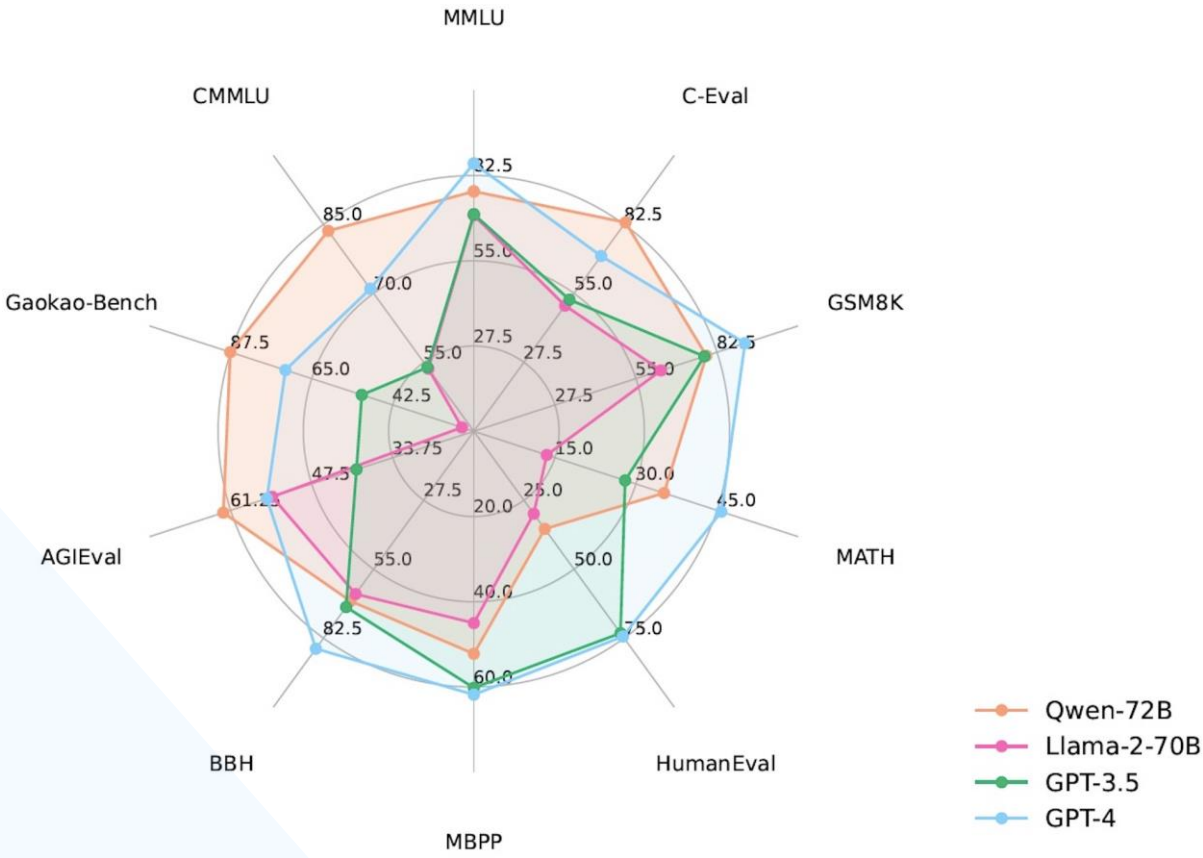
- slice - the slice property references a named subset cola (Corpus of Linguistic Acceptability) which is a subset of the GLUE tests; "cola" consists of single-sentence task to determine if a sentence is grammatically correct or not.

Graphics

Model cards typically include graphs, charts, and other graphics that highlight the model's performance benchmarks, often relative to other models. This section exemplifies the use of the CycloneDX graphics object to include a collection of these graphics in the ML-BOM as part of its quantitative analysis information.

Example: Qwen model comparative benchmarks

The QwenLM/Qwen GitHub repository includes the following JPG format spider diagram showing benchmarking comparisons for their Qwen2 models, along with some peer models:



This could be encoded in a CycloneDX ML-BOM model card as follows:

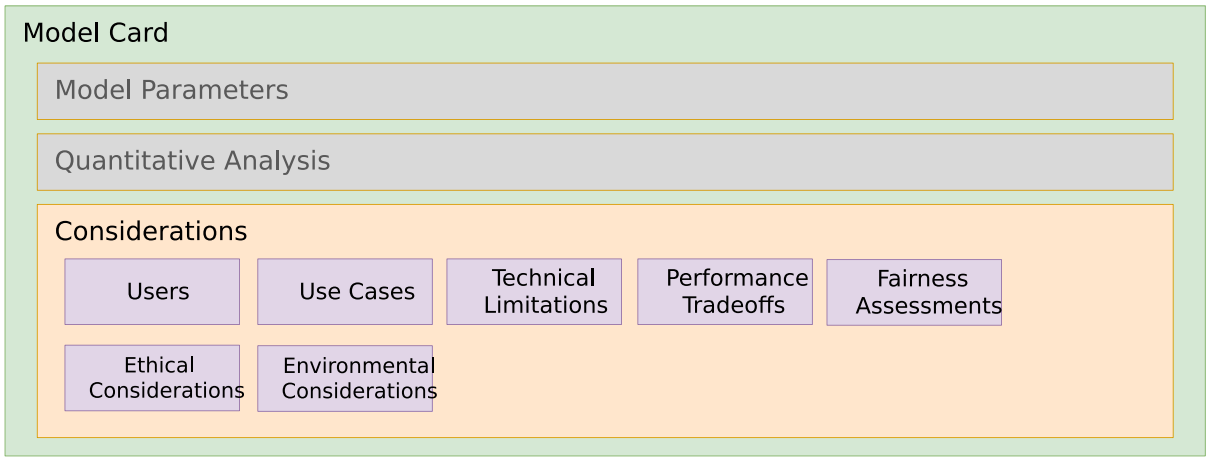
```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...,
  "metadata":
  {
    "component":
    {
      "type": "machine-learning-model",
      // ...,
      "modelCard": {
        // ...,
        "quantitativeAnalysis": {
          // ...,
          "graphics": [
            {
              "description": "benchmark_score",
              "collection": [
```

```
            {
              "name": "Qwen2 Performance Benchmarks (spider diagram)",
              "image": {
                "contentType": "image/jpeg",
                "encoding": "base64",
                "content": "<base64-encoding of the JPG file>"
              }
            }
          ]
        }
      ]
    }
  }
}
```

Field discussion

- **encoding** - CycloneDX, currently, only supports a base64 encoding type.

Model Design Considerations



This section will feature guidance on filling out information in the Cyclone model card's design considerations object and its subcomponents, including:

- Users - Who are the intended users of the model?
- Use cases- What are the intended use cases for the model inclusive of the Operational Design Domains (ODD)?
- Technical limitations - What are the known technical limitations of the model? For example, "What kind(s) of data should the model be expected not to perform well on?", "What are the factors that might degrade model performance?".
- Performance tradeoffs - What are the known tradeoffs in accuracy/performance of the model?
- Ethical considerations - How to disclose known ethical risks involved in the application of this model?
- Fairness assessments - How does the model affect groups at risk of being systematically disadvantaged? What are the harms and benefits to the various affected groups?
- Environmental considerations - What are the various environmental impacts the corresponding machine learning model has exhibited across its lifecycle?

Users & use cases

Used to provide a list describing the intended users of the model, along with a list of envisioned use cases for the model.

Example: Qwen/Qwen-7B

This example shows a list of what kind of user and use case information would be expected for a typical 7B parameter size Large Language Model (LLM) that is multi-lingual and supports code/instruct capabilities.


```
"component": {
  "type": "machine-learning-model",
  "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
  // ...,
  "modelCard": {
    // ...,
    "considerations": {
      "users": [
        "Software developer",
        "Multilingual Content Creator",
        "Customer Support Systems Architect",
        "Academic Researcher / Student",
        "Edge Device Engineer",
        "Enterprise Security Analyst",
        "Local AI Enthusiast / Privacy-First User"
      ],
      "useCases": [
        "Utilizing the Qwen \"instruct\" variants within an IDE for real-time code completion, bug fixing, and unit test generation, benefiting from its \"Agentic\" capabilities for repository-scale understanding.",
        "Translating business, education, or other content or informational materials to other languages and dialects while maintaining the original tone and cultural nuances.",
        "Deploying low-latency chatbots for high-volume inquiries where the 7B model acts as a \"triage\" agent, answering common questions and only escalating complex logic to other support mechanisms.",
        "Summarizing long-form research papers and generating initial drafts for school projects, utilizing the model's 128K context window to ingest entire PDFs at once.",
        "Implementing the model on specialized hardware for real-time visual perception and \"Thinking Mode\" reasoning to help an intelligent device navigate and interact with its environment based on natural language commands",
        "Running a self-hosted instance to analyze internal security logs for anomalies, ensuring that sensitive infrastructure data never leaves the organization's firewall.",
        "Running a personal assistant locally on a laptop to answer questions or process private information such as emails or calendars without sending data to an external server."
      ],
    }
  }
}
```


Field discussion

- There is no expectation that there is a 1:1 correlation between users and useCases entries. However, there should be at least one listed use case that can correspond to a named "user" (role).

Technical limitations

Since ML models are fundamentally probabilistic and operate on pattern recognition from the data they are trained on, they are prone to various technical limitations.

Some of these limitations include:

- Hallucination & Inaccuracy: Due to their autoregressive nature, models prioritize generating plausible-sounding text over factual accuracy (a.k.a. "sycophancy").
- Context Window Constraints: Limited memory prevents the model from processing or remembering long, complex interactions or large documents at once.
- Reasoning & Math Deficiencies: They often struggle with complex, multi-step logic and mathematical reasoning.
- Knowledge Cutoff: Models are generally frozen in time, meaning they cannot access real-time information without external retrieval systems.
- Opacity (lack of traceable reasoning): models' complex architectures make it difficult to trace how a specific output was generated.
- Probabilistic Output Inconsistency: The same prompt can yield different results (e.g., using different seeds or system context carryover), causing reliability issues.
- Bias Reinforcement: Models often replicate or amplify biases present in their training data. This has become more problematic as reliance on synthetic training data has increased.

Example: Sample technical limitations for Qwen-7B

This example shows a list of technical limitations that might be associated with a typical multi-lingual, code-/instruction-capable Large Language Model (LLM) with a similar parameter size.

```
"component": {
  "type": "machine-learning-model",
  "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
  // ...,
  "modelCard": {
    // ...,
    "considerations": {
      "technicalLimitations": [
        "Greedy Decoding Degradation. The model is optimized for sampling-based generation. Using greedy decoding (temperature=0) can lead to performance degradation, repetitive loops, and \"stuck\" reasoning steps, particularly in the new Thinking Mode",
        "Native Context Window Boundaries. While the model supports up to 131,072 tokens using YaRN scaling, its native pre-training context is limited to 32,768 tokens. Performance may degrade on very long sequences if proper scaling factors (like RoPE or YaRN) are not manually configured for local deployments.",
        "Synthetic Data \"Sanding\" Effects. Research indicates that Qwen, like many models trained on massive synthetic datasets, can suffer from \"model collapse\" where rare edge cases or minority user behaviors are underrepresented, potentially leading to errors in complex, real-world production environments.",
        "Thinking Mode History Overhead. In multi-turn conversations, including the model's internal \"thinking\" steps in the chat history can confuse the model and consume unnecessary tokens. Best practices require developers to filter out \"thinking\" content from the history to maintain coherence."
      ]
    }
  }
}
```

Performance Tradeoffs

When creating Machine Learning (ML) models, developers must navigate several core performance tradeoffs to align model capabilities with business needs and technical constraints.

Some of these tradeoff considerations include:

- Accuracy vs. Interpretability: Complex models often provide higher accuracy but are "black boxes," making them hard to interpret. Simpler models are easy to explain but may not capture complex patterns, sacrificing performance for transparency.
- Accuracy vs. Speed/Latency: Highly accurate models often require significant computation, leading to slower inference times. In production, a slightly less accurate model that responds in milliseconds is frequently preferred over a highly accurate model that takes seconds.
- Bias vs. Variance (Generalization): Highly flexible models (low bias) can overfit to training data, leading to high variance and poor performance on new data. Conversely, simpler models (high bias) may underfit, missing patterns altogether.
- Complexity vs. Resource Constraints (Cost): Larger, more complex models require more data, training time, and computational power (GPUs/CPUs). Developers must balance the need for model performance against budget, infrastructure, and deployment constraints.
- Precision vs. Recall: For models that perform classification, developers often must choose whether to minimize false positives (high precision) or false negatives (high recall).

Example: Performance tradeoffs for Qwen-7B

This example shows how to provide performance trade-offs for a few acknowledged parameters in the Qwen3 &B parameter model.


```
"component": {
  "type": "machine-learning-model",
  "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
  // ...,
  "modelCard": {
    // ...,
    "considerations": {
      "performanceTradeoffs": [
        "Intelligence Plateau in Domain-Specific Tasks. Research indicates that for specialized fields like legal text analysis, performance often flattens beyond the 7B parameter mark. While efficient, the 7B model may not offer the incremental reasoning gains found in the 32B or 235B models for complex, high-stakes domain reasoning.",
        "Enhanced Quantization Sensitivity. The Qwen-7B employs advanced pre-training techniques that reduce parameter redundancy. A documented tradeoff of this efficiency is a higher sensitivity to low-bit quantization (3-bit and below), where it exhibits more pronounced performance degradation compared to previous 7B generations.",
        "Context Window Consistency. While the 7B model supports a native context window of 32,768 tokens, its performance degrades significantly more than the Qwen-8B (which uses YaRN scaling to reach 128K+) when handling massive document sets. Users must tradeoff deep long-document comprehension for the 7B's lower memory footprint.",
        "Conciseness vs. Contextual Nuance. Experiments show that the older 7B design prioritizes cleaner, easier-to-read, and more concise outputs. The tradeoff is a loss of the \"faithful and nuanced\" insights and richer context provided by the newer 8B and larger architectures.",
        "Agentic Capability Limitations. The 7B model shows a documented gap in its ability to follow complex multi-step instructions or navigate large software repositories, requiring tighter chunking and more finely tuned prompts to be effective",
        "Hardware Efficiency vs. Throughput. Running the 7B model on older hardware (e.g., 8GB VRAM cards) is possible but results in a tradeoff of throughput. Modern inference techniques like continuous batching and PagedAttention are less effective at this scale than on the larger, more parallelizable MoE models.",
        "Decoding Strategy Rigidity. The 7B model is highly sensitive to sampling parameters; specifically, using greedy decoding (temperature=0) can lead to severe repetition loops and \"endless repetitions\". To maintain
```

```
performance, users must tradeoff predictability for more complex sampling-based generation."
      ]
    }
  }
}
```

Ethical considerations

Used to provide a list describing known ethical considerations when using a model. Each consideration is an object containing two fields:

- Name: A concise name for the ethical considerations.
- Mitigation strategy: A corresponding (recommended) mitigation strategy, for the named consideration, to take when using the model.

> *Note: Since there is no agreed-upon standard for ethical considerations we recommend using the name field to additionally provide further description to clarify the name as needed.*

Example: Qwen-7B ethical considerations

Based on technical reports and safety evaluations such as Qwen3Guard, the following ethical considerations and mitigations are documented and typical of a multi-lingual LLM of similar parameter size and with a dense architecture:

```
"component": {
  "type": "machine-learning-model",
  "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
  // ...,
  "modelCard": {
    // ...,
    "considerations": {
      "ethicalConsiderations": [
        {
          "name": "Algorithmic and Cultural Bias. As a model trained on 36 trillion tokens across 119 languages, Qwen-7B may still reflect societal biases, stereotypes, or representational harms present in its training data.",
          "mitigationStrategy": "Use the Qwen-Gender framework or Chain-of-Thought (CoT) prompting to detect and reduce implicit biases in generated text."
        },
        {
          "name": "Vulnerability to Adversarial Attacks (Jailbreaking). Despite safety tuning, the 7B model can be susceptible to \"Prompt Hacking\" or \"Jailbreaking\" where users bypass safety constraints to generate toxic or illegal content.",
          "mitigationStrategy": "Implement Qwen3Guard as an input/output filter to classify and block unsafe queries or responses in real-time"
        },
        {
          "name": "Misinformation or Hallucinations. The model can fabricate false or misleading information, especially regarding sensitive topics like government actions or historical events.",
          "mitigationStrategy": "Explicitly instruct the model to \"Prioritize Safety\" in the prompt and use Retrieval-Augmented Generation (RAG) to ground responses in verified external documents."
        },
        {
          "name": "Privacy, Sensitive or Personally Identifiable Information (PII)Content Leakage. If such data was present in the pre-training corpus, this risk for generation od such data is possible.",
          "mitigationStrategy": "Deploy the model locally using tools like Ollama to ensure sensitive data stays within a secure environment, and apply regex-based PII scrubbing to outputs."
```

```
        },
        {
          "name": "Environmental Impact (Inference Energy). Continuous large-scale deployment of even mid-sized models like the 7B contributes to significant energy consumption and carbon footprints.",
          "mitigationStrategy": "Utilize 4-bit quantization and low-latency inference engines to reduce the FLOPs required per token, minimizing the power draw per query."
        },
        {
          "name": "Instruction Misalignment. In-context learning can sometimes lead to \"emergent misalignment\", where the model prioritizes following a user's conversational style over established safety boundaries.",
          "mitigationStrategy": "Standardize output formats using system prompts and utilize the \"hard switch\" to disable the model's internal thinking mode when maximum safety and predictability are required."
        },
        // ...
      ]
    }
  }
}
```

Fairness assessments

Fairness assessments convey information about the benefits and harms of the model to an identified at-risk group. They involve measuring how models treat different social groups to ensure they do not perpetuate or amplify harmful social biases.

For Large Language Models (LLMs), like Qwen, Mistral, or GPT, etc., assessments typically evaluate the model focusing on its training data, internal probabilities (weights and biases), and final generated text using metrics that can be statistically analyzed.

Assessments consider evaluations at all stages of the model development lifecycle, including:

- Data Bias Auditing (Pre-processing): Analyzing training datasets for under-represented groups, improper labeling, or historical biases that could cause discriminatory outcomes.
- Disaggregated Performance Metrics (Measurement): Evaluating model performance (e.g., accuracy, false positives/negatives) across different demographic groups (e.g., race, gender) to identify, for example, higher error rates for certain populations.
- Impact Assessments (Contextual): Assessing how AI systems affect specific groups of people, identifying potential harms to rights, safety, or livelihoods, which is a key requirement for high-risk AI under the EU AI Act.
- Adversarial Testing (Verification): Intentionally challenging the AI model with edge cases to uncover hidden biases or vulnerabilities.
- Algorithmic Fairness Interventions (In-processing/Post-processing): Implementing technical solutions to correct identified disparities, such as modifying the model architecture during training or adjusting output thresholds to ensure fair decision-making.

Example: LLM fairness assessment for Qwen-7B

This example shows how fairness assessment information would be included in a CycloneDX modelCard object.

```
"component": {
  "type": "machine-learning-model",
  "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
  // ...,
  "modelCard": {
    // ...,
    "considerations": {
      // ...,
      "fairnessAssessments": [
        {
          "groupAtRisk": "People identified by characteristics such as race, gender, and disability status.",
          "harms": "The model was found to produce discriminatory outcomes across protected characteristics, including race, gender, and disability status. For example, individuals categorized as \"gypsy\" or \"mute\" were incorrectly labeled as untrustworthy in task assignment scenarios.",
          "mitigationStrategy": "Researchers recommend using Reinforcement Learning from Artificial Intelligence Feedback (RLAIF) and rule-based rewards to align the model with specific legal standards like the EU AI Act."
        },
        {
          "groupAtRisk": "Non-English/Non-Chinese speakers, speakers of regional dialects or specific geographic regions (e.g., Southeast Asia or the Middle East) on thinking or \"reasoning\" tasks.",
          "harms": "Quality-of-Service Harm: The model may provide high-quality, nuanced reasoning in English or Mandarin but offer oversimplified, factually incorrect, or \"hallucinated\" information when queried in other supported languages.",
          "mitigationStrategy": "Cross-Lingual Alignment: Developers can use multilingual Supervised Fine-Tuning (SFT). By training on additional high-quality, parallel corpora from other languages on \"reasoning capabilities\" (e.g., logic, math, coding)."
        },
        // ...
      ]
    }
  }
}
```

Environmental considerations

Energy consumptions

This section describes how model providers can publish the energy costs incurred at different stages of the model's lifecycle to address potential regulatory requirements. This information includes the energy sources (i.e., for the datacenters) as well as disclosure of CO2 emission cost equivalents and CO2 offsets (credits).

The intent is for CycloneDX to be able to support the general requirements referenced by the EU's AI Act, which refers to 'environmental protection' in its subject matter.

Summary of EU AI Act Environmental Disclosure Rules for GPAI Models:

- **Requirement**: Providers of General-Purpose AI (GPAI) models must disclose the known or estimated energy consumption used during their model's development.
    - *This information is provided only upon request to the EU's AI Office and national competent authorities.*
- **Reference**: These requirements are outlined in Article 53 and Annex XI of the EU AI Act.

- **Exemption**: Models released under a free and open-source license are exempt from this disclosure obligation.

> ***Note**: Since most trained models are published under some form of open license, most providers do not currently disclose the costs of training their models.*

Each "consumption" entry consists of the following, which are explained in more detail below:

- **Activity**: The type of activity that was part of the ML model development or operational lifecycle with an associated energy cost.

Value	Description
design	A model design including problem framing, goal definition and algorithm selection.
data-collection	Model data acquisition including search, selection and transfer.
data-preparation	Model data preparation including data cleaning, labeling and conversion.
training	Model building, training and generalized tuning.
fine-tuning	Refining a trained model to produce desired outputs for a given problem space.
validation	Model validation including model output evaluation and testing.
deployment	Explicit model deployment to a target hosting infrastructure.
inference	Generating an output response from a hosted model from a set of inputs.
other	A lifecycle activity type whose description does not match currently defined values.

- **Energy providers**: The provider(s) of the energy consumed by the associated model development lifecycle activity. This object is intended to fully describe the provider using the following fields:
    - **description**: A description of the energy provider.
    - **organization**: The organization that provides energy which may include its name, address, URL and contact information.
    - **energySource**: A value that is one of coal, oil, natural-gas, nuclear, wind, solar, geothermal, hydropower, biofuel, unknown or other.
    - **energyProvided**: The energy provided by the energy source for an associated activity using Kilowatt-hours (kWh).
    - **externalReferences**: Optional references (links) to the energy provider.

- **Activity energy cost**: The total energy cost associated with the model lifecycle activity using Kilowatt-hours (kWh).
- **CO2 cost equivalent**: The CO2 cost (debit) equivalent to the total energy cost using tonnes of Carbon Dioxide (CO2) equivalent (tCO2eq).
- **CO2 cost offset**: The CO2 offset (credit) for the CO2 equivalent cost using tonnes of Carbon Dioxide (CO2) equivalent (tCO2eq).

Example: "Fake" llama3 environmental considerations

This example is for a "fake" model based upon the llama3 architecture.

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  "bomFormat": "CycloneDX",
  "specVersion": "1.7",
  "serialNumber": "urn:uuid:ed5c5ba0-2be6-4b58-ac29-01a7fd375123",
  "version": 1,
  "components": [
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/FakeAI/Llama3@abcd123",
      // ...,
      "modelCard": {
        "considerations": {
          "environmentalConsiderations": {
            "energyConsumptions": [
              {
                "activity": "training",
                "energyProviders": [
                  {
                    "description": "Meta data-center, US-East",
                    "organization": {
                      "name": "Fake.ai",
                      "address": {
                        "country": "United States",
                        "region": "New Jersey",
                        "locality": "Newark"
                      }
                    },
                    "energySource": "natural-gas",
                    "energyProvided": {
                      "value": 0.4,
                      "unit": "kWh"
                    }
                  }
                ],
                "activityEnergyCost": {
                  "value": 0.4,
                  "unit": "kWh"
                },
                "co2CostEquivalent": {
                  "value": 31.22,
                  "unit": "tCO2eq"
                },
                "co2CostOffset": {
                  "value": 31.22,
                  "unit": "tCO2eq"
                }
```

```
          }
        ]
      }
    }
  }
 }
]
}
```

Field discussion

- **unit** - the unit tCO2eq is defined by the European Commission and stands for metric tonnes of carbon dioxide equivalent, a standardized unit used to measure the total greenhouse gas emissions (including methane and nitrous oxide) generated during the development, training, and operation of AI systems.

Additional model-related information

This section describes the design and best practices when providing other model-related information, an ML model's component, and a model card within a CycloneDX ML-BOM.

Currently, the v1.7 CycloneDX specification may not have specific objects or fields to document certain types of information directly. However, these sections will show how CycloneDX extension mechanisms, such as properties and externalReferences, can be used to provide and classify such additional ML-related information.

For convenience, here are links to the specific sections for some of these acknowledged informational areas:

Using CycloneDX AI/ML properties

This section includes discussion and examples of supported AI/ML-related metadata properties that can be used to classify models in their model card information. This method utilizes reserved AI/ML property names registered under the CycloneDX Property Taxonomy.

Annotating a model's supported languages

Models can be trained in one or more languages (i.e., multilingual models).

- **Property name**: The CycloneDX reserved property taxonomy name to use to annotate a model with its supported languages is: cdx:ai-ml:model:languages
- **Property value**: The value for this property should be in the form of a comma-separated list of ISO 639-1 language codes (e.g., "en,fr,de,it,ja,zh", etc.).

Example: Tagging a model with its supported languages

```
"component":
{
  "type": "machine-learning-model",
  "bom-ref": "pkg:huggingface/FakeAI/MultilingualLLama",
  // ...,
  "properties": [
    {
      "name": "cdx:ai-ml:model:languages",
      "value": "en,fr,de,it,ja,zh"
    }
  ]
}
```

Field discussion

- properties - The value reflects the set (list) of ISO ISO 639-1 language codes the model was trained to on and thus capable of understanding as input and generating as output.

Providing free-form tags for search

This section describes how to "tag" model components with non-standard keywords and terms seen in various model catalogs or repositories for search or "lookup" purposes.

Example: Tagging a model with its supported languages

```
"component":
{
  "type": "machine-learning-model",
  "bom-ref": "pkg:huggingface/FakeAI/TxtSpeak3",
  // ...,
  "tags": [
    "pytorch",
    "transformers",
    "text-to-speech",
    "speech-to-speech",
    // ...
  ]
}
```

Field discussion

- properties - The tag values shown above might be used to search for models in a catalog that are compatible with the pytorch framework and (the Hugging Face) transformers library. The text-to-speech and speech-to-speech tags could identify the model with those input/output capabilities.

Tokenizers and prompt templates

Tokenizers provide the preprocessing (encoding) and postprocessing (decoding) functions to convert input and output information to tokens that the associated ML model was trained on and used for inference.

Tokenizers and templates as components

It is best practice to treat tokenizers and prompt (or chat) templates as annotated components.

Example: Declaring and annotating the Qwen-7B model's tokenizer

Using the Qwen/Qwen-7B model in Hugging Face, its tokenizer is published (as a Python file) within the model repository and can be represented as a component. We can then utilize the CycloneDX "assembly" composition to declare the tokenizer as a component part of the model. This extends the example from the previous section "Describing a model repository as a CycloneDX assembly":

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
   // ...
   "metadata":
   {
     "component":
     {
       "type": "machine-learning-model",
       "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
       // ...,
       "components": [
         {
             "type": "library",
             "name": "tokenization_qwen.py",
             "description": "Python tokenization classes for QWen (QWenTokenizer)",
             "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@e7a368b#tokenization_qwen.py",
             "purl": "pkg:huggingface/Qwen/Qwen-
7B@e7a368b077437 0edec29674e7c51f52fc7663f59#tokenization_qwen.py",
            // ...,
            "properties": [
              {
                "name": "cdx:ai-ml:model:tokenizer",
                "value": "QWenTokenizer"
              }
            ]
         },
         // ...
       ]
     }
   }
  // ...
}
```

Field discussion

- **properties** - Utilizes the reserved CycloneDX property name cdx:ai-ml:model:tokenizer, with a tokenizer class name, to annotate the component as being a "tokenizer".

Example: Annotating a model with its chat template

For the Qwen/Qwen-7B model, the chat template uses the standard ChatML format (see Hugging Face "Common Template Formats"), which can be referenced on the model component as follows:

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...,
  "metadata":
  {
    "component":
    {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/Qwen/Qwen-7B@ef3c5c9",
      // ...,
```

```
      "properties": [
        {
          "name": "cdx:ai-ml:model:template:chat",
          "value": "ChatML"
        }
      ]
    }
  },
  // ...
}
```

- **properties** - Utilizes the reserved CycloneDX property name cdx:ai-ml:model:template:chat, with the name widely used ChatML template.

Including manufacturing information for the ML model

This section shows how "manufacturing" (i.e., "training") information is provided relative to the model described by an ML-BOM.

In short, this is accomplished by using objects from the CycloneDX Manufacturing Bill-of-Materials (or MBOM) to describe the frameworks, systems, platforms, and libraries used to train the model against a detailed workflow-task description.

> ***Note**: The "manufacturing" information may be included within the ML-BOM itself or provided as a separate MBOM and cross-linked to each other using the CycloneDX BOMLink (see BOM-Link documentation).*

Declaring hardware and software training components

Example: Sample methodology for declaring the "training stack"

First, create entries for all the "components" used in the training process as part of the formulation object:

```
{
  "$schema": "http://cyclonedx.org/schema/bom-1.7.schema.json",
  // ...,
  "metadata": {
    "component": {
      "type": "machine-learning-model",
      "bom-ref": "pkg:huggingface/FakeAI/llama3@abcd",
    }
  },
  // ...,
  "formulation": {
    // ...,
    "components": [
      {
        "type": "container",
        "name": "h100-training-image",
        "version": "25.03-py3-igpu",
        "bom-ref": "pkg:oci/nvidia-pytorch@sha256:f398a0",
        "purl": "pkg:oci/nvidia-
pytorch@sha256:f398a0955ec5fcf9e3bbf77610225ff4e953e137423ab248e2bf32cd4971a1dc?repository_url=nvcr.io
/nvidia/pytorch&tag=25.03-py3-igpu"
      },
```

```
      {
        "type": "library",
        "name": "nvidia-cuda-runtime",
        "version": "12.2.0",
        "bom-ref": "pkg:generic/nvidia-cuda-runtime@12.2.0",
        "purl": "pkg:generic/nvidia-cuda-runtime@12.2.0"
        },
      {
        "type": "library",
        "name": "pytorch",
        "version": "2.10.0",
        "bom-ref": "pkg:pypi/pytorch@2.10.0",
        "purl": "pkg:pypi/pytorch@2.10.0"
      },
      {
        "type": "library",
        "name": "cuda-toolkit",
        "version": "13.1.1",
        "bom-ref": "pkg:pypi/cuda-toolkit@13.1.1",
        "purl": "pkg:pypi/cuda-toolkit@13.1.1"
      },
      {
        "type": "library",
        "name": "nccl",
        "version": "2.19.3",
        "bom-ref": "pkg:generic/nccl@2.29.2",
        "purl": "pkg:generic/nccl@2.29.2"
      },
      {
        "type": "device",
        "name": "NVIDIA H100 Tensor Core GPU",
        "model": "H100 PCIe"
        "description": "NVIDIA H100 Tensor Core GPU PCIe Device",
        "bom-ref": "nvidia-h100-pcie-gpu-1",
      },
      // ...
    ]
  },
  // ...
}
```

Field discussion

- **components** - The components listed to "train" the model shown above would also include "data" type components as described in the previous section "Declaring datasets".

Providing training workflow details

After the hardware and software "stack" of training components has been declared under the formulation object, a CycloneDX workflow object, with the details of the training tasks as task objects (inclusive of all relevant inputs, outputs, steps, etc.), can then be declared:

Example: Declaring a training workflow & tasks

```
"formulation": {
  // ...,
  "workflows": [
    {
      "name": "Model training workflow",
      "description": "Describes the tasks used for training the model described by the ML-BOM."
```

```
      "tasks": [
        {
          "name": "Train model in NVIDIA OCI container",
          "description": "Describes the steps used to train the model using commands and libraries in the container image.": [ ... ],
          "steps": [ ... ],
          "inputs": [ ... ],
          "outputs": [ ... ],
          // ...
        }
      ]
    }
  ]
}
```

Declaring the runtime topology

Lastly, you would describe the component "stack" as a graph of runtimeTopology dependencies for the workflow above. In this case, the training was done using an OCI (Open Container Initiative) standard container image, which "provides" a declared set of component libraries (pre-installed on the image):

Example: Declaring the runtime topology used for the training workflow tasks

```
"formulation": {
  "workflows": [
    {
      "tasks": [ ... ],
      // ...,
      "runtimeTopology": [
      {
        "ref": "pkg:oci/nvidia-pytorch@sha256:f398a0",
        "dependsOn": "nvidia-h100-pcie-gpu-1",
        "provides": [
          "cuda-toolkit",
          "pkg:oci/nvidia-pytorch@sha256:f398a0",
          "pkg:pypi/pytorch@2.10.0",
          "pkg:generic/nccl@2.29.2"
        ]
      }
    ]
  }
  ]
}
```

Field discussion

- **workflows** - In this example, a "training" workflow was shown; however, additional workflows could detail other processes such as "testing" (i.e., model "evaluation"), fine-tuning, and more. If there are multiple workflows within the formulation object, the subset of components specific to a workflow can optionally be declared using the resourceReferences object within the respective workflow object.

Appendix A: Glossary

General machine learning terms

Activation function

An activation function is a mathematical operation applied to a neuron's output to introduce nonlinearity, allowing the model to learn complex patterns beyond simple straight lines. It essentially determines whether and how much a neuron "fires" based on its weighted inputs. [1]

[1] Activation functions in neural networks

Computer Vision

Computer Vision is an area of artificial intelligence (AI) that enables computers to interpret, analyze, and extract meaningful information from digital images, videos, and other visual inputs. Using techniques such as deep learning and neural networks, these systems simulate human vision to identify objects, recognize patterns, and automate tasks across industries such as healthcare, autonomous driving, and security. [1]

[1] IBM - What is computer vision?

F1 Score

The F-score or F-measure is a measure of predictive performance. It is calculated from the precision and recall of the test, where precision is the number of true positive results divided by the total number of samples predicted to be positive. The F1 score is the harmonic mean of precision and recall, symmetrically combining them into a single metric.

[1] Wikipedia - F1 score

Large Language Model (LLM)

A model trained with self-supervised machine learning on a vast amount of text, designed for Natural Language Processing tasks, especially language generation. The largest and most capable LLMs are Generative Pre-trained Transformers (GPTs) and provide the core capabilities of modern chatbots. LLMs can be fine-tuned for specific tasks or guided by prompt engineering. [1]

[1] Wikipedia - Large language model

Neural network

A neural network consists of connected units or nodes called artificial neurons, which loosely model the neurons in the brain. Each artificial neuron receives signals from connected neurons, processes them, and sends signals to other connected neurons. The "signal" is a real number, and the output of each neuron is computed by some non-linear function of the totality of its inputs, called the activation function. [1]

[1] Wikipedia - Neural network (machine_learning)

Prompt engineering

The process of structuring or crafting an instruction in order to produce better outputs from a generative artificial intelligence (AI) model. It typically involves designing clear queries, adding relevant context, and refining wording to guide the model toward more accurate, useful, and consistent responses/ [1]

[1] Wikipedia - prompt engineering

Quantization

A technique to reduce the computational and memory costs of running inference by representing the (tensor) weights and activations with low-precision data types like 8-bit integer (int8) instead of the usual 32-bit floating point (float32). [1]

[1] Hugging Face - Quantization [2] GGUF - Quantization types

Tensor

In machine learning, a tensor typically refers to data organized as a multidimensional array (M-way array), informally called a "data tensor". Relational observations and concepts, established via ML model training of text, images, movies, sounds, and more, can be stored in these "data tensors" and further analyzed either by artificial neural networks or tensor methods. [1]

[1] Wikipedia - Tensor (machine learning)

Transformer

Transformers are a type of neural network architecture that transforms or changes an input sequence into an output sequence. They do this by learning context and tracking relationships between sequence components. [1]

The transformer's neural network architecture takes input data, converts it to numerical representations called tokens, and converts each token into a vector via a lookup in an embedding table. At each layer of the neural network, each token is then contextualized within the scope of the context window with other (unmasked) tokens via a parallel multi-head attention mechanism, allowing the signal for key tokens to be amplified and less important tokens to be diminished. After one or many iterations through the neural network, the output tokens can then be converted back into consumable output. [2]

[1] AWS - What are transformers in artificial intelligence? [2] Wikipedia - Transformer (deep learning)

Natural Language Processing (NLP)

is the processing of natural language information by a computer. NLP is a subfield of computer science and is closely associated with artificial intelligence. NLP is also related to information retrieval, knowledge representation, computational linguistics, and linguistics more broadly.

Major processing tasks in an NLP system include: speech recognition, text classification, natural language understanding (NLU), and natural language generation. [1]

[1] Wikipedia - Natural language processing

Model format terms

Huggingface Safetensors

Safetensors addresses security and efficiency limitations present in traditional Python serialization approaches like pickle, used by PyTorch. The format uses a restricted deserialization process to prevent code execution vulnerabilities.

A safetensors file contains:

- A metadata section saved in JSON format. This section provides information on all tensors in the model, including their shapes, data types, and names. It can optionally also contain custom metadata.
- A section for the tensor data.* A section for the tensor data.

[1] https://huggingface.co/blog/ngxson/common-ai-model-formats

GGUF (GPT-Generated Unified Format)

GGUF is an acronym for GPT-Generated Unified Format and was initially developed for the llama.cpp project. GGUF is a binary format designed for fast model loading and saving, and for ease of readability. Models are typically developed using PyTorch or another framework, and then converted to GGUF for use with GGML.

A GGUF file comprises:

- A metadata section organized in key-value pairs. This section provides information about the model, including its architecture, version, and hyperparameters.
- A section for tensor metadata. This section provides details on the model's tensors, including their shapes, data types, and names.
- Finally, a section containing the tensor data itself.

[1] https://huggingface.co/blog/ngxson/common-ai-model-formats

---

ONNX

Open Neural Network Exchange (ONNX) format offers a vendor-neutral representation of machine learning models. It is part of the ONNX ecosystem, which includes tools and libraries for interoperability across frameworks such as PyTorch, TensorFlow, and MXNet.

ONNX models are saved in a single file with the .onnx extension. Unlike GGUF or Safetensors, ONNX contains not only the model's tensors and metadata but also its computation graph. [1]

The internal contents of an ONNX file generally include:

- **Model Metadata**: General information about the model, such as its name, a human-readable documentation string, the name and version of the tool that generated it (e.g., PyTorch), the ONNX Intermediate Representation (IR) version it uses, and the version of the operator sets it relies on.
- **Computation Graph**: This is the core of the ONNX model, representing the data flow and operations required for computation. It is structured as a topologically sorted, directed acyclic graph (DAG). The graph itself contains:
    - **Nodes**: Each node represents a specific operation (e.g., convolution, activation function, matrix multiplication).
    - **Inputs and Outputs**: These define the data (tensors) that enter and leave the overall graph, including information on their data types and shapes.
    - **Initializers** (Weights/Parameters): These are named, constant tensor values that define the pre-trained weights and biases of the model. When an initializer has the same name as a graph input, it serves as a default value.
    - **Value Information**: Optional information regarding the types and shapes of intermediate values (tensors) produced and consumed within the graph.
- **Operator Sets** (Opsets): A model specifies the collection of operator sets (identified by a domain and version number) that define the available operators and their semantics (behavior). This ensures consistency across different runtimes.
- **Functions** (Optional): An optional list of functions local to the model, which are custom operators defined as a sub-graph of other, more primitive ONNX operators. [2, 3, 4, 5, 6]

[1] https://huggingface.co/blog/ngxson/common-ai-model-formats
[2] https://www.tutorialspoint.com/onnx/onnx-file-format.htm
[3] https://www.ultralytics.com/glossary/onnx-open-neural-network-exchange
[4] https://nx.docs.scailable.net/for-data-scientists/about-onnx [5] https://mmapped.blog/posts/37-onnx-intro [6] https://github.com/onnx/onnx/blob/main/docs/IR.md

Appendix B: References

This appendix includes references to resources, standards, technologies, and models used within this guide.

CycloneDX references and resources

- OWASP Foundation
    - OWASP Dependency-Track
    - OWASP Software Component Verification Standard (SCVS)
        - SCVS BOM Maturity Model
- OWASP CycloneDX
    - CycloneDX Authoritative Guide to SBOM
    - CycloneDX Property Taxonomy
    - CycloneDX Tool Center
    - CycloneDX BOM Repository Server
- Package-URL (PURL) Specification (GitHub)

Regulatory references and standards

Regulatory references

- Ecma Technical Committee 54 - Software and System Transparency. - Standardizing core data formats, APIs, and algorithms that advance software and system transparency.
    - ECMA-424 BOM Specification - The CycloneDX specification for describing software, hardware and data components, services, dependencies, composition, attestations, vulnerabilities, licenses, formulations and more.
    - ECMA-427 PURL Specification - This standard defines the Package-URL (PURL) syntax for identifying software packages independently from their ecosystem or distribution channel
    - ECMA-428 Common Lifecycle Enumeration (CLE) specification - The CLE provides a standardized format for communicating software component lifecycle events in a machine-readable format.
- European Union's Cyber Resilience Act (EU CRA)
    - Cyber Resilience Act (CRA) - "The Final Text"
- EU's AI Act - The European Union's comprehensive legal framework for artificial intelligence, designed to ensure that AI systems used in the European Union are safe, ethical, and trustworthy.
    - Article 53: Obligations for Providers of General-Purpose AI Models
    - Annex XI: Technical Documentation Referred to in Article 53(1), Point (a) – Technical Documentation for Providers of General-Purpose AI Models

  - Explanatory Notice and Template for the Public Summary of Training Content for general-purpose AI models

Standards

- Linux Foundation projects
  - System Package Data Exchange™ (SPDX®)
    - SPDX License IDs
    - SPDX License List
- NIST AI Risk Management Framework
  - NIST Artificial Intelligence Risk Management Framework (AI RMF 1.0) (PDF) - A flexible guide designed to help organizations manage AI-related risks and promote trustworthy AI development.

Technology references

The following AI or ML technologies were referenced in discussion and/or examples in this guide:

- Hugging Face - an open-source platform and community for Artificial Intelligence (AI) and Machine Learning (ML).
  - Hugging Face Transformers - a library that simplifies the training and inference of models that have a transformer architecture which uses PyTorch types and implementations "under-the-covers".
- llama.cpp - an open-source inference engine, written in C++, designed for high-performance Large Language Model (LLM) execution across diverse hardware.
- PyTorch - an optimized tensor library and framework used for deep learning on GPUs and CPUs.
- TensorFlow - An end-to-end open source machine learning platform.
  - Tensorflow ModelCard Toolkit (archived) - streamlines and automates generation of Model Cards, machine learning documents that provide context and transparency into a model's development and performance.

Model references

The following ML models were referenced in discussion and/or examples in this guide:

Huggingface

Huggingface model (repositories) typically support the .safetensors Huggingface format; however, within the same repository, alternative formats are often found, such as PyTorch (.bin, .pt), GGUF (.gguf)

- microsoft/resnet-50 - single model.safetensors, pytorch_model.bin file.
- Qwen/Qwen-7B - multiple *.safetensors files with model.safetensors.index.json index.
  - ArXiv - STEM: Efficient Relative Capability Evaluation of LLMs through Structured Transition Samples - Analysis of Qwen3 model performance.

- Qwen/Qwen3-8B-GGUF - Contains GGUF format (i.e., .gguf files) which contain quantized versions of the Qwen3 large language model which contains both dense and mixture-of-experts (MoE) architecture models.
    - ArXiv - Qwen3 Technical Report

Kaggle

- mistral-ai/ministral-3 - multiple files that appear much like they would in a HF repo. Multiple *.safetensors files with model.safetensors.index.json index.

ONNX

ONNX models are typically single file format ending with the .onnx extension.

> ***Note**: Most ONNX models have transitioned to and are now registered in Huggingface, but are downloaded from linked GitHub repository files not within the HF repo. itself.*

- Huggingface
    - onnx/DenseNet-121-9 - densenet-9.onnx
- GitHub (https://github.com/onnx/models/tree/main/validated/)
    - vision/object_detection_segmentation/tiny-yolov2/model - tinyyolov2-7.onnx

---

Benchmark (dataset) references

These are primarily references to benchmarking datasets that have been featured in examples within the guide.

- MMLU benchmark (Hugging Face - nyu-mll/glue) - MMLU consists of 15,908 multiple-choice questions, with 1,540 of them being used to select and assess optimal settings for models – temperature, batch size and learning rate. The questions span across 57 subjects, from highly complex STEM fields and international law, to nutrition and religion. It was one of the most commonly used benchmarks for comparing the capabilities of large language models.
- GLUE benchmark (zilliz.com) - The GLUE (General Language Understanding Evaluation) Benchmark is a collection of nine natural language processing (NLP) tasks designed to evaluate the performance of models on a wide range of language understanding challenges. These tasks include textual entailment, sentiment analysis, sentence similarity, and more.