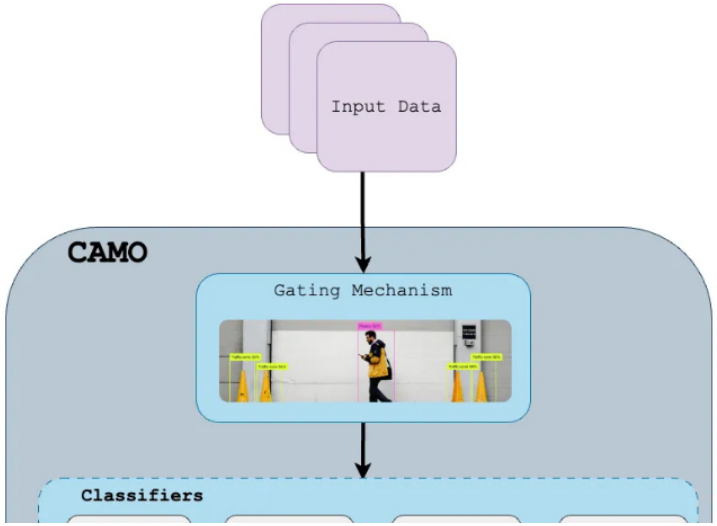
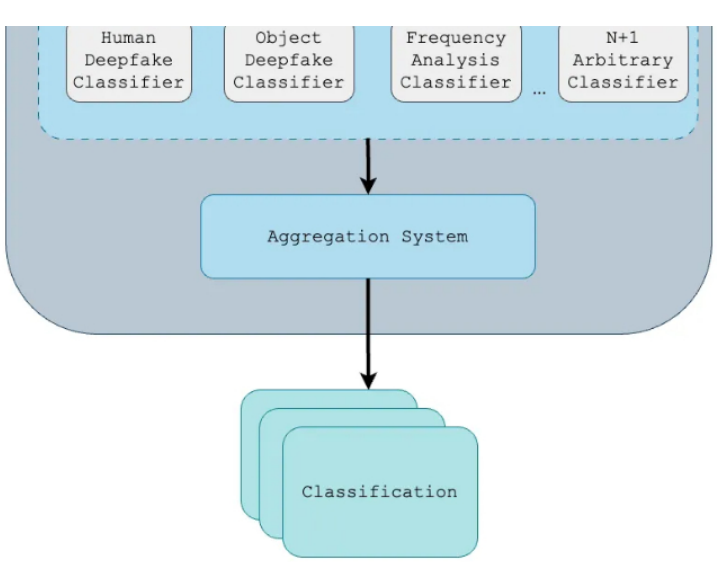
CAMO: Content-Aware Model Orchestration (CAMO) Framework for Deepfake Detection

We're excited to announce the release of the initial version of our Content-Aware Model Orchestration (CAMO) framework for deepfake detection. In this post, we'll discuss the motivation for developing this framework, showcase the novel features of its design, and introduce several open source datasets and pre-trained generalist and expert models. We'll also explore the future directions we are considering.

Highlights

- Novel hard mixture-of-experts framework leveraging generalist and specialist/expert models for deepfake detection
- Modular image generation pipeline designed to produce synthetic datasets with semantic balance
- Open source datasets created by the aforementioned pipeline, containing synthetic data that semantically mirror open source real image datasets
- Reproducible with open-source training code, datasets, and pre-trained weights
- GitHub repository: **bitmind-subnet**
- HuggingFace models and datasets: https://huggingface.co/bitmind





Why Content Aware Model Orchestration?

As deepfakes grow more sophisticated, traditional single-model detection techniques fall short. With different generative algorithms leaving behind nuanced, unique forgery artifacts, state-of-the-art models have been shown to generalize poorly to outputs of unseen models. Our CAMO framework tackles this challenge by leveraging a hierarchy of expert models, each attuned to different aspects of deepfake detection. This strategy allows for specialized models to learn **content-specific forgery features** without being burdened by the expectation to generalize to different content. It also **improves interpretability** through insights into the decision-making process, and **offers adaptability** by allowing easy integration of new expert models as deepfake methods evolve.

Improving Scalability and Complexity

The CAMO system's architecture excels in scalability over traditional monolithic approaches by breaking down deepfake detection into more manageable subproblems, each solved by a specialized model. Content-aware routing during inference dynamically allocates resources for selected downstream models as needed, based on the specific input. This design offers the following benefits:

1. **Parallel Processing** - Various gating functions operate asynchronously, selecting the appropriate models to invoke based on specific criteria.
2. **Subproblem Specialization** - CAMO breaks down deepfake detection into manageable subproblems, enabling specialized models to excel within defined constraints. This approach avoids the need for models to generalize beyond their intended expertise.
3. **Modular, extensible architecture** - CAMO's architecture allows contributors to easily integrate or update individual expert models, addressing the latest deepfake challenges without needing to retrain the entire system. Additionally, its detection and gating mechanism is designed to be readily adaptable to new categories.

Theoretical Backing

Our approach is supported both by general computer vision research as well as recent work on deepfake detection.

1. Mixture of Experts for Face Forgery Detection: This study validates the use of multiple expert models for detecting face forgeries.
2. Hierarchical Fake Image Detection: Presented at ECCV 2022, this paper proposes a hierarchical method for fake image detection.
3. Hard Mixtures of Experts for Large Scale Weakly Supervised Vision: This paper demonstrates the feasibility of training far larger models than could be practically trained with standard CNN architectures.
4. UCF: Uncovering Common Features for Generalizable Deepfake Detection: These authors created the UCF architecture which similarly aims to disentangle the deepfake forgery method feature space for improved generalizability.

Approach: CAMO Framework Architecture

Our CAMO deepfake detection system combines multiple specialized deepfake detectors in a **hard mixture of experts (HMoE) structure**, allowing for both fine-grained analysis and high-level decision making.

HMoE differs from the more traditional (soft) mixture of experts in that it does not blend outputs from every single expert model - rather, the hard routing selectively invokes the best model(s) for the job. While HMoE is naturally more efficient than traditional MoE (only a fraction of the parameters are used for any one prediction), we are exploring ways to introduce soft routing where outputs from multiple experts would lead to improved accuracy at the cost of efficiency.

Method Overview

1. Implemented a **generalized hard mixture of experts framework** capable of routing images to specific models and combining outputs in a configurable fashion.
2. Implemented a **gating mechanism** for orchestration of models in the hierarchy
3. Developed **specialized detectors** for different aspects of deepfake detection, trained on content-specific datasets of real and synthetic images, to act as expert models in the hierarchy
4. Generated **semantically balanced synthetic datasets** with our custom image generation pipeline. These datasets act as synthetic "mirrors" of open source real image datasets, ensuring an equal distribution of topics across the two halves of our training datasets.

Initializing Experts

Upon initialization, CAMO sets up a YOLOv8x object detector as part of its gating mechanism, chosen for its speed and performance. It also initializes separate models for `'face'` and `'general'` deepfake detection, which are activated selectively based on outputs of the gating mechanism. In the following example, detectors are loaded with our finetuned UCF model weights using a `load_detector` function using the adapted UCF backbone.

```
self.detectors = {}

self.detectors['face'] = self.load_detector(
    UCF_CONFIG_PATH,
    UCF_WEIGHTS_PATH,
    BITMIND_FACES_CHECKPOINT_NAME,
    predictor_path,
    'face'
)

self.detectors['general'] = self.load_detector(
    UCF_CONFIG_PATH,
    UCF_WEIGHTS_PATH
```

```
    BITMIND_CHECKPOINT_NAME,
    predictor_path,
    'general'
)

self.object_detector = YOLO("yolov8x.pt")
```

Initializing CAMO's expert deepfake and object detection models.

Gating Mechanism

Images are asynchronously classified using YOLOv8 and a dlib face detector, identifying content based on detected classes and confidence levels. The system is designed for extensibility, allowing custom handling of diverse image contents.

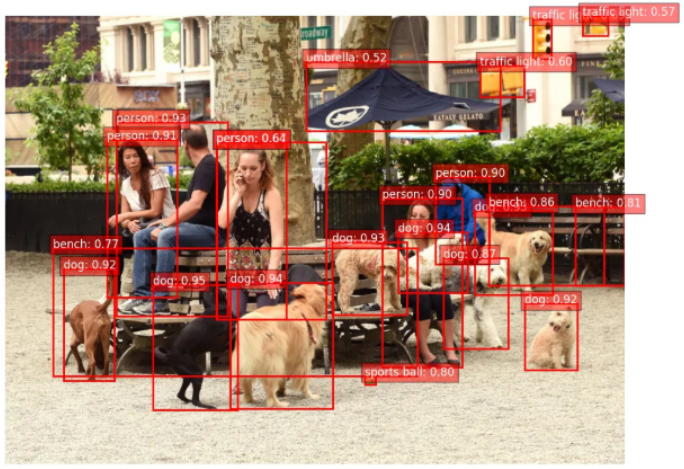
Future improvements to CAMO could involve adjusting the ratio of bounding box sizes to image dimensions to prioritize classifications more effectively. Grouping similar objects (e.g., `'zebras'`, `'dogs'`, `'cats'`) under a single category (e.g., `'animals'`) could also streamline the classification process.

```
async def classify_image(self, image, use_object_detection=True):
    faces, num_faces = self.detectors['face'].detect_faces(image)

    if use_object_detection:
        results = self.object_detector(image)
        detected_classes = [
            result.names[box.cls.item()]
            for result in results
            for box in result.boxes if box.conf.item() > 0.5
        ]
        if 'person' in detected_classes and num_faces:
            return 'face', faces

    return 'face', faces if num_faces else ('general', None)
```

Asynchronous image classification with optional object detection using a YOLOv8 model.



YOLOV8x object detector bounding boxes on a dog park.

Another approach we are exploring involves delegating the analysis of individual bounding boxes to specialized models; i.e. a patch-wise strategy for deepfake detection. This method may be particularly effective in scenarios where synthetic content is confined to a small section of an image, like a face.

Hard Mixture-of-Experts Inference

Once incoming images are processed by the gating mechanism, image data are routed to the appropriate expert model. As with the previous functions, this method is designed with extensibility and interpretability in mind.

```
image_bytes = base64.b64decode(synapse.image)
image = Image.open(io.BytesIO(image_bytes))

image_type, faces = await self.classify_image(image)

if image_type == "face":
    image_tensor = self.detectors['face'].preprocess(image, faces=faces)
    pred = self.detectors['face'].infer(image_tensor)
else:
    image_tensor = self.detectors['general'].preprocess(image)
    pred = self.detectors['general'].infer(image_tensor)

synapse.prediction = pred

return synapse
```

Delegation of image data to expert models.

Data

Training Data

To train generalist and expert models for CAMO, we build on a foundation of curated real and fake image datasets publicized by leading machine vision research teams at NVIDIA, Microsoft, Google, and others. We use datasets reputable in image segmentation, recognition, and classification spaces.

Additionally, we introduce:

1. **Novel deepfake image datasets using our synthetic image generation pipeline**

> *While capable of generating datasets offline, our synthetic image generation pipeline also runs on our Bittensor subnet's validators, which are constantly producing new data*

2. **Tailored, preprocessed image datasets for training expert models**

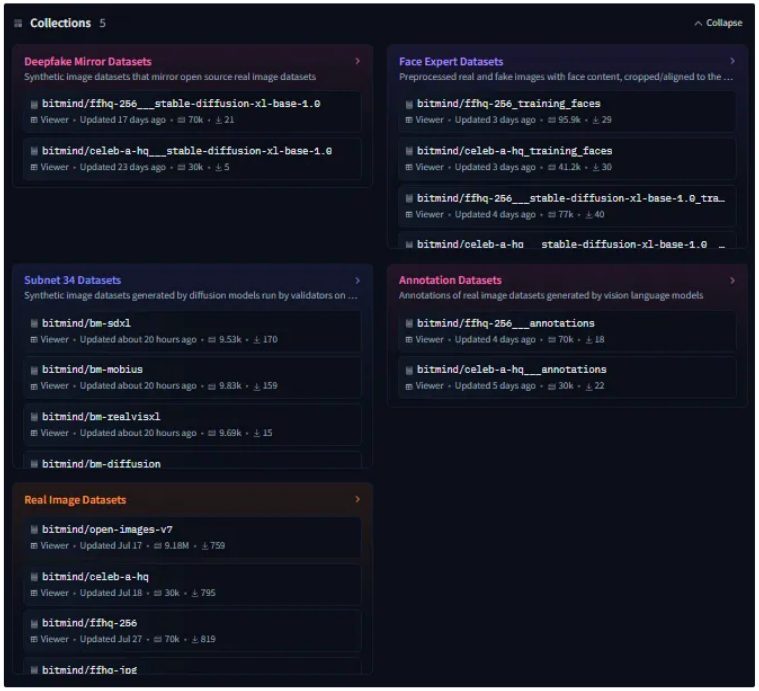
> *As datasets for content-specific training jobs can be computationally expensive to create, we provide fully preprocessed datasets and associated pipelines in our HuggingFace and GitHub*

During generalist model training, datasets are augmented using random image transformations to improve data diversity, including crop, resize, and rotation operations. These operations are already applied as part of the preprocessing pipeline for expert dataset creation.

1. Generalist Datasets

This combination of curated and in-house generated datasets feature diverse image content, and are open sourced on BitMind's HuggingFace. Included are real images from Open Images, FFHQ, and Celeb-A-HQ, and deepfake images created with our synthetic image generation pipeline to mirror the real datasets.

We are constantly reviewing our dataset collections—Our goal is to offer comprehensive data to create the most robust generalist and expert deepfake detectors.



BitMind's real and deepfake image datasets are continuing to grow, with new preprocessed expert training datasets and continuous additions to image diversity.

2. Expert Datasets

Our image datasets for training expert face deepfake detectors are released on HuggingFace in our Face Expert Datasets collection. These contain images from real and fake datasets of face images that have undergone a preprocessing procedure at scale:

1. Filtering for face content with a pretrained face detector.
2. Cropping and aligning to landmarks of the largest detected face.
3. Perturbation using different image transformation operations.

Below is the abstracted batch processing snippet from our `TrainingDatasetProcessor` class used to create and upload preprocessed training datasets:

```
class TrainingDatasetProcessor:
  ...
  def preprocess_faces_only(self, dataset_dict, transform, batch_size=32):
      for split in dataset_dict.keys():
          dataloader = DataLoader(dataset_dict[split].with_format("torch"),
                                  batch_size=batch_size,
                                  shuffle=False)
```

```
        preprocessed_dataset = {"image": [],
                                "original_index": [],
                                "landmark": [],
                                "mask": []}

        for batch_idx, batch in tqdm(enumerate(dataloader), total=len(dataloader)):
            images = batch['image']
            # Convert tensors to PIL images and apply transform
            images = self._apply_transform_to_images(images, transform)
            # Use pretrained dlib models to crop and align to face landmarks
            valid_faces_present, valid_faces, valid_landmarks, valid_masks = self._proc
            if any(valid_faces_present):
                self._update_preprocessed_dataset(preprocessed_dataset, valid_faces_pre

        # Replace dataset with filtered, cropped, and aligned face data
        dataset_dict[split] = Dataset.from_dict(preprocessed_dataset)
```

The resulting datasets are partitioned by transform-specific subsets, to allow AI engineers the flexibility of creating training splits with different transformations.

Balanced Synthetic Image Dataset Generation

To ensure our synthetic image datasets match the balanced feature set of our real image datasets, we create images that closely "mirror" our real images on a near 1:1 correspondence. We have developed a synthetic image generation pipeline featuring SOTA models for **image-to-caption translation, text moderation, and image generation**.

The process begins by captioning real images, followed by moderation of these captions by a large language model (LLM). The moderated captions are then used by a diffusion model to generate synthetic images. This integration has developed a robust vision-to-language and text-to-image model system for generating diverse, realistic images to train deepfake detectors. Our goals are:

- **Enhance dataset diversity** and **generate high-quality images rapidly** for improved model generalizability.
- **Utilize synthetic outputs** for scoring on our Bittensor subnet.
- **Publish all code and datasets** on the BitMind GitHub and BitMind Huggingface repositories for transparency with the Bittensor community and as contribution to the broader open source community.
- **Maintain model output latency** under 1 minute.

1. Image Captioning: BLIP-2, OPT-6.7b, fine-tuned on COCO

We selected the 6.7B parameter BLIP-2 model, fine-tuned on COCO, for its SOTA image captioning capabilities. Developed in 2023, it is not only open-source but also computationally efficient, minimizing trainable parameters during pre-training. This model stands out as it delivers superior performance across a wide array of benchmarks on Paperswithcode, making it ideal for our needs.

```
description = ""
prompts = ["An image of", "The setting is", "The background is",
           "The image type/style is"]

# Generating image descriptions using prompts
for i, prompt in enumerate(prompts):
    description += prompt + ' '
    inputs = self.processor(image, text=description, return_tensors="pt").to
                            (self.device, torch.float16)
    generated_ids = self.model.generate(**inputs, max_new_tokens=max_new_tokens)
```

```
    answer = self.processor.batch_decode(generated_ids,
                                         skip_special_tokens=True)[0].strip()
if answer:
    description += answer + ' '
else:
    description = description[:-len(prompt) - 1]

if self.apply_moderation:
    moderated_description = self.moderate_description(description)
    return moderated_description

return description.strip() + '.'
```

Image caption generation

2. Moderation: Meta-Llama-3.1-8B-Instruct-bnb-4bit (Unsloth)

We selected Unsloth's ungated version of Meta's Llama 3.1 8B-Instruct LLM to moderate captions generated by the BLIP2 model based on its standing as the top-performing lightweight model (under 70B parameters) in the IFEval category on the HF Open LLM Leaderboard. While other metrics like BBH and MuSR were considered, this LLM's high IFEval, which evaluates a model's ability to adhere to explicit formatting instructions, ensures that our moderation processes meet stringent standards for accuracy and reliability.

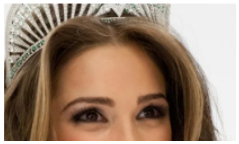
We use the following prompt for moderation:

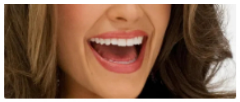
> "You always concisely rephrase given description, eliminate redundancy, and remove all specific references to individuals by name."

```
moderated_text = self.text_moderation_pipeline(
    messages,
    max_new_tokens=max_new_tokens,
    pad_token_id=self.text_moderation_pipeline.tokenizer.eos_token_id,
    return_full_text=False
)
```

Image caption moderation

Below are before-and-after comparisons to showcase how image captions improve with moderation.

Real Image	BLIP-2 Caption	LLM Moderation
	A picture of a baby laying down on a bedThe setting is a blue roomThe background is a white wall	A baby lies on a blue blanket in a sunny setting, surrounded by a blue background, in a portrait view.
	A picture of a woman wearing a tiara and smilingThe setting is a pageantThe background is a white background	A beauty pageant winner, a Colombian representative, poses with a smile on a stage with a white background in a portrait-



style image.

A picture of a beautiful blonde woman with blue eyesThe setting is a studioThe background is a white background

A woman in a studio setting, wearing a white tank top and sporting blonde hair, is photographed in a portrait style against a white background.

3. Synthetic Image Generation with Diffusion Models

For improved organization and modularity, we define diffusion models in a `constants.py` file. These models are dynamically loaded into our `SyntheticImageGenerator` class, enabling easy extension of our image generation functions with pipeline-specific features like image dimensions.

Python

Copy

```python
VALIDATOR_MODEL_META = {
    "diffusers": [
        {
            "path": "stabilityai/stable-diffusion-xl-base-1.0",
            "use_safetensors": True,
            "variant": "fp16",
            "pipeline": "StableDiffusionXLPipeline"
        },
    ]
}

DIFFUSER_ARGS = {
    m['path']: {k: v for k, v in m.items() if k != 'path' and k != 'pipeline'}
    for m in VALIDATOR_MODEL_META['diffusers']
}

DIFFUSER_PIPELINE = {
    m['path']: m['pipeline'] for m in VALIDATOR_MODEL_META['diffusers'] if
    'pipeline' in m
}

DIFFUSER_NAMES = list(DIFFUSER_ARGS.keys())
```

Example of defining diffusion model constants for dynamic loading

An upcoming update will introduce optional generation parameters such as `guidance_scale` and `num_inference_steps` to support our integration of Flux models.

Results

CAMO was evaluated against several benchmark model architectures, all trained on the same dataset to offer a fair comparison of performance. We deployed the models described below on our subnet as miners and monitored their accuracy.

1. **BitMind-NPR** is our original base miner, prior to the development of CAMO. The architecture comes from a 2024 CVPR paper by Tan et al., and employs a traditional ResNet50 architecture along with a novel Neighboring Pixel Relationships metric used to guide training towards

learning localized artifacts specific to the upsampling operations common to many generative algorithms.

2. **DeepfakeBench-UCF** is the top performing model on the DeepfakeBench leaderboard, with the pretrained weights supplied by the authors. The purpose of including this model is to provide a baseline that demonstrates the lack of generalization capabilities of traditional pre-trained open source models.
3. **BitMind-UCF** is the same UCF architecture from DeepfakeBench, fine-tuned on datasets specific to our subnet. The purpose of including this model is both to demonstrate the power of fine-tuning, and provide a performant baseline against which to compare CAMO.
4. **CAMO** is our Content Aware Model Orchestration framework, which here leverages two specialized UCF backbones. CAMO demonstrates significant improvements over all 3 baselines.

Online SN 34 Performance

Model	Accuracy	Precision	Recall	F-1
BitMind-NPR	74.75	83.05	54.44	65.77
DeepfakeBench-UCF	66.48	73.51	44.76	55.64
BitMind-UCF	82.24	81.03	81.50	81.27
CAMO	**87.65**	**88.64**	**88.14**	**88.34**

Breakthrough Insight: Content-Specific Expert Models

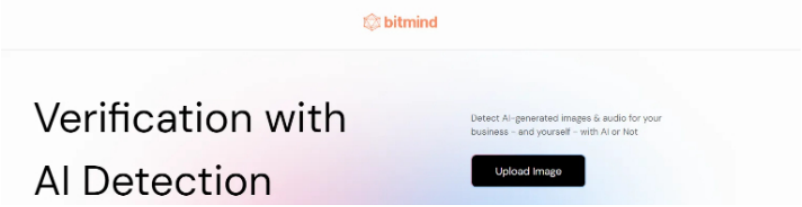
During the development of the CAMO model, we made a significant discovery that has far-reaching implications for the field of deepfake detection: the power of leveraging a combination of content-specific expert models in improving detection accuracy.

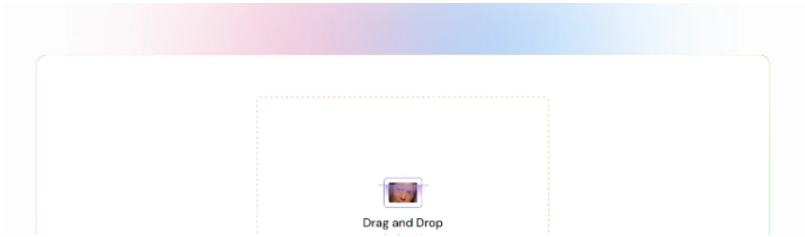
The Limitation of One-Size-Fits-All Approaches

Since the breakthrough of ChatGPT, the approach to improving AI models has been "Bigger Model x More Data." However, in the realm of deepfake detection, we've found that this method has limitations. The entropy of digital content and forgery fingerprints is extremely high, and there are inherent limits to traditional training techniques that make it challenging for a single, large model to effectively handle all types of digital content.

Aligning with Current Academic Research

Our findings align with the state-of-the-art in current academic research, where studies often focus on specific types of content. For example, DeepfakeBench (DFB) scopes their benchmark in detecting manipulated or generated faces. These specialized approaches have shown impressive results within the constraints of their respective domains. The goal of CAMO is to offer these constraints in a comprehensive system that allows specialized models to thrive without being required to generalize beyond their expertise.





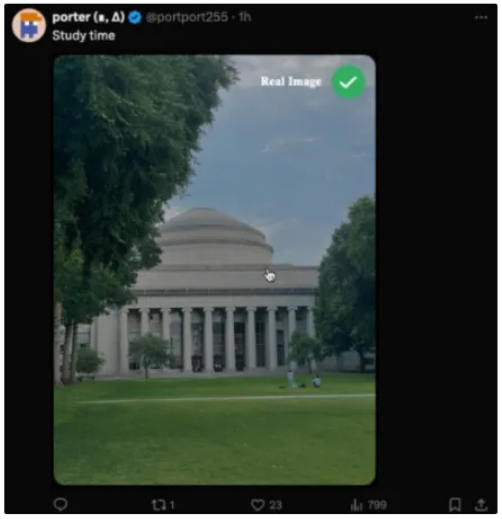
A BitMind web application using subnet APIs to detect uploaded deepfake images with miner models

Real-World Impact

In practice, this breakthrough has allowed us to achieve detection accuracies that surpass those of larger, monolithic models, especially when dealing with diverse types of digital content.

While the majority of academic research remains siloed in archived GitHub repositories, we've created several consumer-facing applications to realize real world-impact from our efforts.

To our knowledge, we're the only group actively building such products while open sourcing our models, code and homegrown datasets.



A **BitMind browser extension** using subnet APIs to detect a fake image during real time web browsing.

Open Source Commitment

We've open-sourced our entire pipeline, including:

1. Training scripts
2. Data generation code
3. Data preprocessing code
4. Model architecture
5. Evaluation procedures

Future Directions

Structured Data Generation for Future Models

As deepfake technology evolves, so too must our detection methods. We're investing in the development of advanced structured data generation techniques to create more diverse and challenging training datasets. This initiative involves:

1. Synthetic Mirror Dataset Generation: By generating synthetic data that mimics real-world semantics we can improve our model's robustness to diverse real-world scenarios.
2. Adversarial Data Generation: We're investigating methods for adversarially generating/modifying deepfakes specifically to avoid detection, towards creating increasingly challenging datasets that push our detection models to their limits.
3. Multi-Modal Data Synthesis: We're exploring ways to generate coherent multi-modal data (e.g., synchronized audio-visual fakes) to train our models on more complex, real-world deepfake scenarios.

This structured approach to data generation will enable us to stay ahead of emerging deepfake techniques and continually improve our detection capabilities.

Expand and Enhance Content Detectors

Our current model excels at detecting a wide range of deepfakes, but we're not stopping there. We're actively working on expanding and enhancing our content detectors to cover an even broader spectrum of manipulated media:

1. Cross-Modal Consistency Checking: We're developing detectors that analyze the consistency between different modalities (e.g., lip sync with audio, body movement with speech patterns) to catch more sophisticated multi-modal deepfakes.
2. Temporal Anomaly Detection: By enhancing our ability to analyze video sequences over time, we aim to catch subtle inconsistencies that may not be apparent in single frames.
3. Fine-grained Manipulation Detection: We're working on detectors that can not only identify the presence of manipulation but also pinpoint the specific areas or aspects of the content that have been altered.
4. Context-Aware Detection: By incorporating broader contextual information (e.g., metadata, distribution patterns, source analysis), we aim to improve our model's ability to distinguish between authentic and manipulated content.

Aggregation and Soft Routing

We are exploring more dynamic, flexible architectures for leveraging the intelligence of individual experts within CAMO. This effort focuses on the gating mechanism and aggregation components of CAMO. Directions we're considering are:

1. Patch-wise delegation of individual bounding boxes to specialized models. This method is particularly effective in scenarios where synthetic content is confined to a small section of an image, like a face.
2. Switching out the gating mechanism and aggregation layer for a gradient boosted tree (GBT) classifier, which are known to be performant in combining the outputs of multiple learners. Traditionally, GBTs are used to combine weak learners, so this may not end up being the right approach given we have full-sized experts, but is worth exploring to test this hypothesis.

By focusing on these three key areas - structured data generation, additional expert detector models, and liquid hierarchy structures - we're positioning our CAMO model to remain at the forefront of deepfake detection technology. We're excited about the potential of these advancements to make significant strides in combating the spread of misinformation through synthetic media and protecting the integrity of digital content.

Join Us in Combating Deepfakes

BitMind is at the frontier of deepfake detection AI research. Our CAMO model represents a significant step forward in the fight against deepfakes. We're committed to advancing this technology and invite researchers, developers, and organizations to:

1. Explore our open-source code, datasets, and pre-trained models
2. Contribute to our ongoing research
3. Collaborate on developing even more robust deepfake detection methods

To learn more about our work and get involved, visit our website bitmindlabs.ai or reach out to our team.