Objectif

- Finetuner le modèle pré-entraîné pour améliorer la génération de mesh texturée.

Contraintes

Données

- Meshes 3D texturés requis
- Dataset limité par crédit
- Peu de données open-source

Copyright

- Difficulté avec IP (jeux vidéo/superhéro)

Modèle

- Faible généralisation concepts récents (fortnite / mr beast)
- Meilleur sur concepts génériques

Entraînement

- Batch size = 1 (6 vues)
- Ressources limitées par limite de quota (Disco)
- Data augmentation nécessaire

Outils et technologies

Génération 3D

- Meshy (texturé, limité)
- Hunyuan3D-2 (non texturé)
- Hunyuan3D-2.1 (texturé, quota)

Dataset

- MV-Adapter pipeline
- RGB, normales, positions
- Ortho10View

Modèle

- MV-Adapter
- DreamShaper XL
- SDXL VAE (fp16)
- PyTorch Lightning

Infrastructure & scripting

- Orchestration avec **Slurm**
- Entraînement distribué sur serveur **Disco**
- Collaboration technique avec **Marc Pignat** pour libérer des quotas
- Mise en local de Hunyuan3D-2.1 pour ne plus avoir de quota

Observations clés

- Concepts connus ⟶ amélioration rapide.
- Concepts peu représentés ⟶ besoin élevé en données.
- Meilleures performances sur catégories génériques non protégées.

Scalabilité

La pipeline mesh puis multiview puis training est facilement extensible. L'utilisation de Slurm permet d'automatiser les runs et de scaler les expérimentations.

Apprentissage

Le modèle dépend fortement de ses connaissances initiales. Avec peu de données, il apprend des éléments simples (comme les couleurs) mais pas des concepts complexes.

Décision

Le finetuning actuel n'est pas suffisant pour une utilisation en production. Il est nécessaire de :

- augmenter fortement la taille du dataset,
- améliorer la qualité des meshes et des textures,
- affiner la stratégie de finetuning.

0.1. Vue d'ensemble

L'objectif est d'avoir une compréhension globale des exigences afin d'entraîner notre propre adaptateur personnalisé.

0.1.1. Aperçu du dataset

L'architecture du dataset est présentée sous la forme suivante :

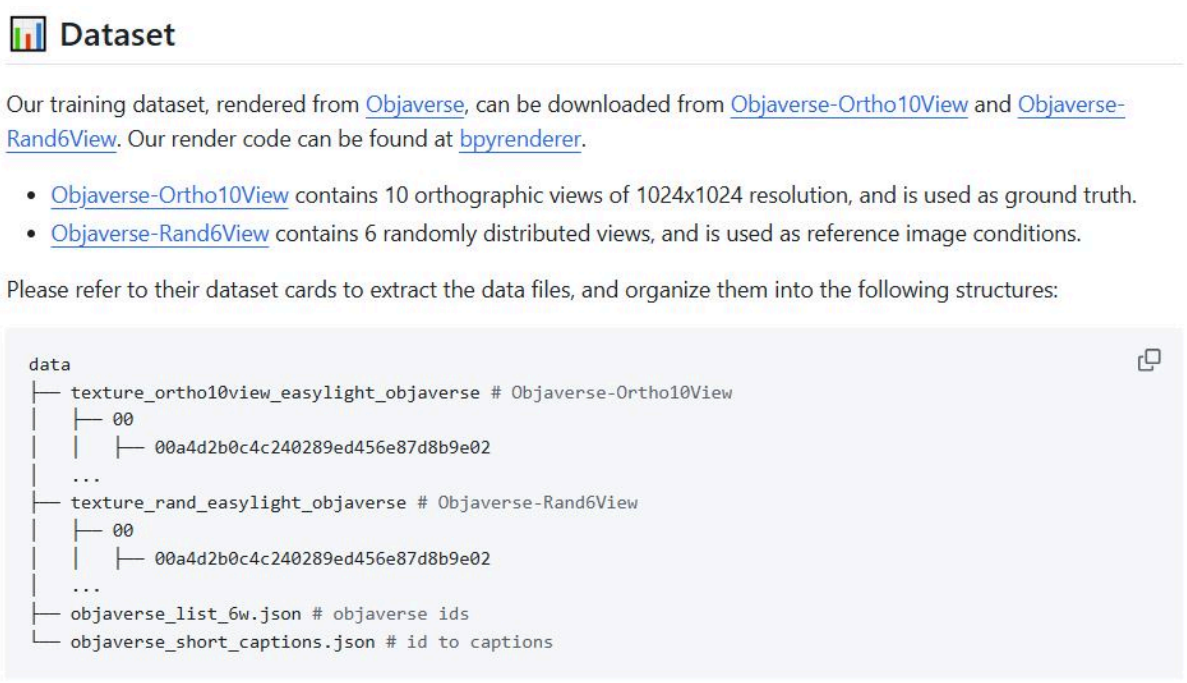


Figure 1: dataset

Il s'agit d'un dossier contenant un dossier **reference** ainsi que des fichiers JSON. Le dossier **reference** contient des images générées.

Il existe deux types de datasets fournis.

Ortho10View contient 10 vues fixes, toujours les mêmes. Il comprend 10 rendus colorés fixes du maillage.

Figure 2: dataset ortho

Rand6View contient 6 vues aléatoires.

```
Objaverse-Rand6View
├── data
│   ├── 00
│   │   ├── 00a4d2b0c4c240289ed456e87d8b9e02
│   │   │   ├── color_0000.webp # rgb
│   │   │   ├── color_0001.webp
│   │   │   ├── ...
│   │   │   ├── color_0005.webp
│   │   │   ├── depth_0000.exr # depth
│   │   │   ├── depth_0001.exr
│   │   │   ├── ...
│   │   │   ├── depth_0005.exr
│   │   │   ├── normal_0000.webp # normal
│   │   │   ├── normal_0001.webp
│   │   │   ├── ...
│   │   │   ├── normal_0005.webp
│   │   │   └── meta.json # camera
│   │   ...
│   │   └── 00a6e74d8eeb428bb3e01b9361d8d57e
│   ├── 0a
│   ├── ...
│   └── zS
├── objaverse_list_6w.json # id list
└── objaverse_short_captions.json # captions
```

Figure 3: dataset rand6view

La principale différence est que, d'un objet à l'autre, les vues de Rand6View sont aléatoires, ce qui le rend adapté au conditionnement par image. En revanche, Ortho10View possède des vues fixes, ce qui le rend adapté au conditionnement de texture.

0.2. Exécution

Voici le code par défaut.

```yaml
data_cls: mvadapter.data.multiview.MultiviewDataModule
data:
  root_dir: data/Ortho10View/data
  scene_list: data/Ortho10View/list_6w.json
  background_color: gray
  image_names: ["0000", "0001", "0002", "0003", "0004", "0005"]
  image_modality: color
  num_views: 6

  prompt_db_path: data/Ortho10View/captions.json
  return_prompt: true

  projection_type: ORTHO

  source_image_modality: ["position", "normal"]
  position_offset: 0.5
  position_scale: 1.0

  train_indices: [0, -8]
  val_indices: [-8, null]
  test_indices: [-8, null]

  height: 768
  width: 768

  batch_size: 1
  num_workers: 16)


system_cls: mvadapter.systems.mvadapter_text_sdxl.MVAdapterTextSDXLSystem
```

```yaml
system:
  check_train_every_n_steps: 1000
  cleanup_after_validation_step: true
  cleanup_after_test_step: true

  # Model / Adapter
  pretrained_model_name_or_path: "Lykon/dreamshaper-xl-1-0"
  pretrained_vae_name_or_path: "madebyollin/sdxl-vae-fp16-fix"
  pretrained_adapter_name_or_path: "/home/marco.caporizzi/MV-Adapter_mine/weight/
mvadapter_tg2mv_sdxl.safetensors"
  init_adapter_kwargs:
    # Multi-view adapter
    self_attn_processor:
"mvadapter.models.attention_processor.DecoupledMVRowColSelfAttnProcessor2_0"
    # Condition encoder
    cond_in_channels: 6
    # For training
    copy_attn_weights: true
    zero_init_module_keys: ["to_out_mv"]

  # Training
  train_cond_encoder: true
  trainable_modules: ["_mv"]
  prompt_drop_prob: 0.1
  image_drop_prob: 0.1
  cond_drop_prob: 0.1

  # Noise sampler
  shift_noise: true
  shift_noise_mode: interpolated
  shift_noise_scale: 8

  # Evaluation
  eval_seed: 42
  eval_num_inference_steps: 30
  eval_guidance_scale: 3.0
  eval_height: ${data.height}
  eval_width: ${data.width}

  # optimizer definition
  # you can set different learning rates separately for each group of parameters, but
note that if you do this you should specify EVERY trainable parameters
  optimizer:
    name: AdamW
    args:
      lr: 5e-5
      betas: [0.9, 0.999]
      weight_decay: 0.01
    params:
      cond_encoder:
        lr: 5e-5
      unet:
        lr: 5e-5

  scheduler:
    name: SequentialLR
    interval: step
    schedulers:
```

```
      - name: LinearLR
        interval: step
        args:
          start_factor: 1e-6
          end_factor: 1.0
          total_iters: 2000
      - name: ConstantLR
        interval: step
        args:
          factor: 1.0
          total_iters: 9999999
    milestones: [2000]

trainer:
  max_epochs: 10
  log_every_n_steps: 10
  num_sanity_val_steps: 1
  val_check_interval: null # changed this to not hit
  check_val_every_n_epoch: 1
  enable_progress_bar: true
  precision: bf16-mixed
  gradient_clip_val: 1.0
  strategy: ddp
  accumulate_grad_batches: 1

checkpoint:
  save_last: true # whether to save at each validation time
  save_top_k: -1
  every_n_epochs: 9999 # do not save at all for debug purpose
```

1. Problem analysis

Notre génération de canard est basé sur des poids de modèles pré-entrainés.

On souhaiterais dans un premier temps finetune notre modèle afin qu'il soit très bon sur la génération de canard.

On a identifié que le modèle pouvait être finetune sur le point suivant: génération de texture copyrighted.

Les données avec une IP active sont difficiles à trouver pour les jeux-video par example elles sont toujours actives. En effet, lors de notre batch d'execution, les résultats étaient bien plus concluant sur les héro que sur les jeux-vidéo. Ceci peut s'expliquer par le fait que les ip de super-héro sont plus vieille que les données de jeux-video qui sont probablement active. C'est pourquoi il est difficile de générer un canard mario ou un canard sonic plutôt qu'un canard batman ou superman.

C'est pourquoi l'un de nos angles d'attaque est de pouvoir nourrir de modele de données qui sont copyright.

1.1. Génération des données

Afin de pouvoir entraîner notre modèle, l'entrainement de MV-Adapter nous oblige à avoir des meshs texturés.

C'est pour cela que dans une première phase, il faut trouver une solution de génération de meshs texturés qui puissent être gratuites et réutilisation.

1.1.1. Meshy

Une solution est de se tourner vers [Meshy](https://www.meshy.ai/fr/discover) qui permet de la génération de mesh et de texture.

Il y a 100 crédits par mois pour le gratuit.

Le nombre de téléchargement à 10 par mois.

1.1.2. Hunyuan3D-2: High Resolution Textured 3D Assets Generation

Ici, une solution sur un espace hugginsface.

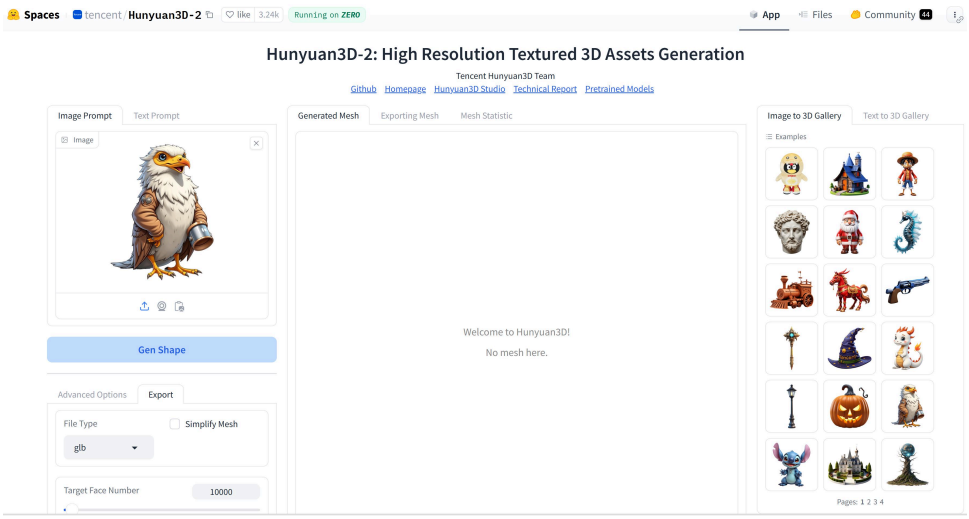
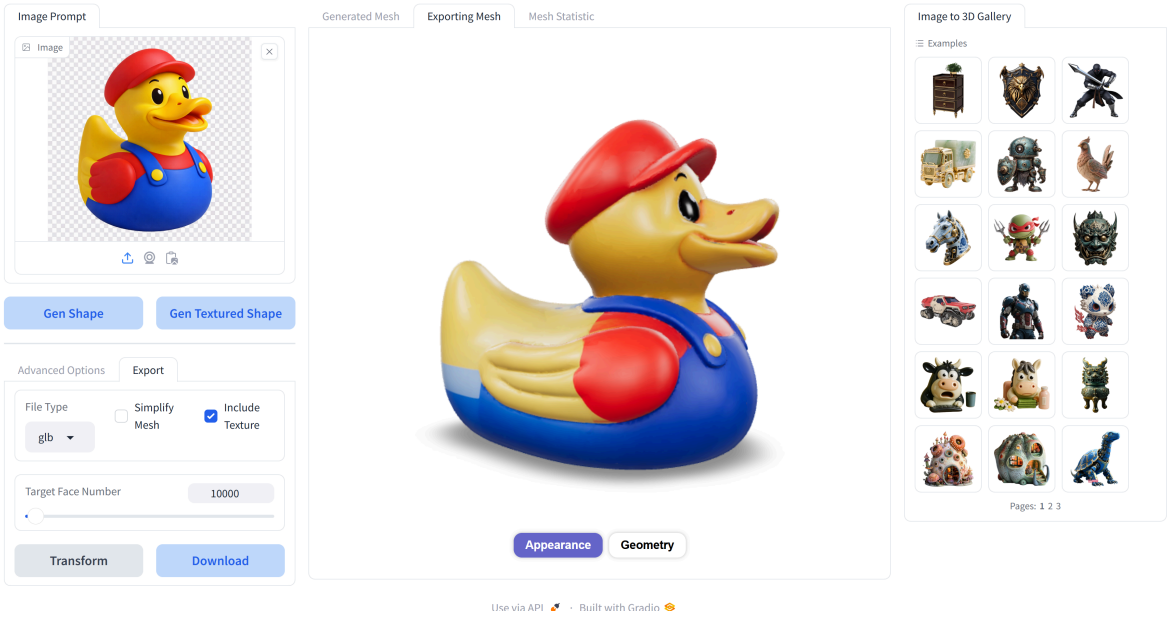


Figure 4: dataset ortho

Mais le mesh est généré sans texture.

1.1.3. Hunyuan-3D-2.1

Ici, le modèle plus récent de Hunyuan de chez tencent qui permet jusqu'à 10 mesh texturées par jour avec la version pro à 9chf par mois.



1.1.4. Décision pour la génération de texture

On préfère épuiser les crédits gratuit de meshy et de Hunyuan3D-2 afin de collecter le plus de données possibles.

Suite à la mise en pause du space, une solution plus axé sur la production a été mise en place, du temps a été pris pour la mise en place de Hunyuan3D-2 avec l'execution sur disco. Ceci est une solution gratuite car elle n'utilise pas de crédit.

1.2. Objectif du finetuning

Le modèle sait générer des superhéros et personnages de jeux-vidéos, le modèle sait générer des canards mais il ne sait pas générer des canards superhéro ou canards jeux-videos.

1.2.1. Création dataset itération sur les personnages de jeux-video

Un des premiers aspect qu'on souhaitait améliorer est la génération d'image copyright. Par exemple, des canards de jeux video: mario, bowser, samus, kratos, sonic.

Cependant, le modèle pré-entrainé n'a pas de résultat concluant de base.

Dans le cas d'un canard mario, l'idée d'un canard avec une casquette rouge et une salopette bleu n'est pas facile à générer.

C'est pour cela, on préfère se diriger vers une génération déjà satisfaisant comme les superhéro.

https://hessoit-my.sharepoint.com/:x:/g/personal/marco_caporizz_hes-so_ch/IQDrMgm9IqLhQ6puCEYr6sdQAXm-EBmT_VTb7w8GNZKG67A?e=BeQ7Bx

1.2.2. Conclusion local

L'une des contraintes rencontré est de bien définir les capacités du modèles pré-entrainé: si le modèle ne génère pas des images copyrightés facilement (mario, sonic), la phase de finetuning devra comprendre la phase d'apprentissage d'un mario et d'un sonic. Ceci implique plus de données d'apprentissage que pour un concept que le modèle sait déjà généré.

1.2.3. Création du dataset itération sur les superhéros

Les générations de canard sur les superhéro sont les plus réussis. Les ip sont anciennes, il y a plus de données et les logos sont réussis.

Cependant, les données copyrightés ont été exclus car les prompts donnée à notre modèle de diffusion (GPT image) était sous copyright donc la génération était annulée.

1.2.4. Conclusion local

Un dataset ne doit pas contenir de copyright pour assurer la reproduction.

1.2.5. Création du dataset itération sur les métiers

Les canards pompiers, de chantier ou astronaute ne sont pas soumis à copyright.

Leur générations pour les modèles pré-entraîné est satisfaisante, comme montrer dans la figure 5,6,7.

{"prompt":"astronaut duck, detailed space suit, reflective glass helmet, oxygen backpack, space environment, white suit with blue accents, stylized but clean, sci-fi

Figure 5: Astronaute model pré-trained

{"prompt":"firefighter duck, red helmet, protective suit with reflective stripes, heavy jacket, gloves, boots, holding fire hose, stylized but clean, rescue worker

Figure 6: Pompier model pré-trained

{"prompt":"construction worker duck, yellow hard hat, orange safety vest with reflective stripes, blue work clothes, gloves, boots, holding tools, stylized but clean, worker

Figure 7: Chantier model pré-trained

Alors afin de récolter des données, j'ai généré les meshs correspondantes.

Figure 8: Astronaute Hunyuan Textured Mesh Generator

Figure 9: Pompier Hunyuan Textured Mesh Generator

Figure 10: Chantier Hunyuan Textured Mesh Generator

Puis j'ai passé chaque meshs texturés dans le multiview creator pour le training.

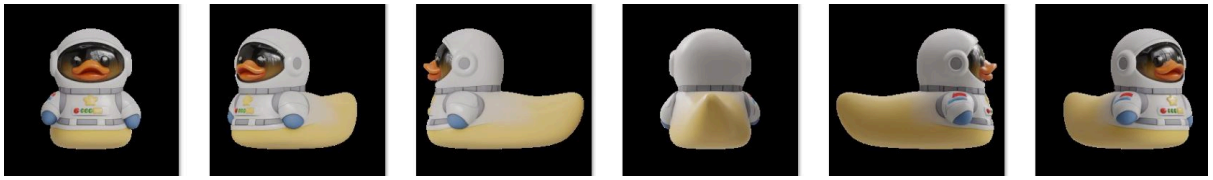


Figure 11: Astronaute MV-Adapter Multiview Generator

Figure 12: Pompier MV-Adapter Multiview Generator

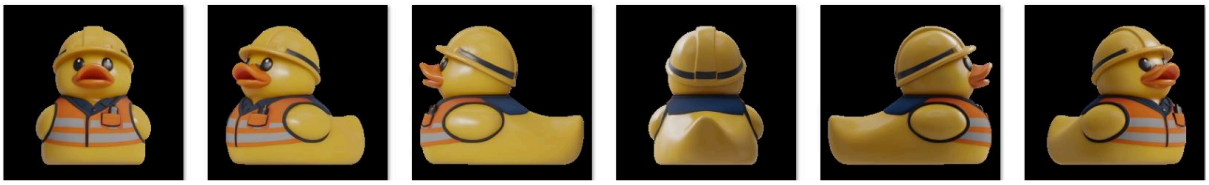


Figure 13: Chantier MV-Adapter Multiview Generator

la normal et la position du canard en 3d.

1.2.6. Finetuning params

Mon set est divisé en 80% pour le training, 10% pour la validation et 10% pour le test.

Mon batch est de 1 mais il comprend toujours 6 views.

Lors de la phase de validation de l'entraînement, on remarque que le modèle génère des canards différents, on suppose qu'il apprend.

1.3. Finetuning

Le dataset est maintenant composé de 200 canards de textures différentes.

Le but est maintenant d'entrainer le modèle pré-entrainé sur les multiview des canards présentés.

1.4. Inference

Le canard chantier:

Figure 14: Inference avec prompt chantier gen1

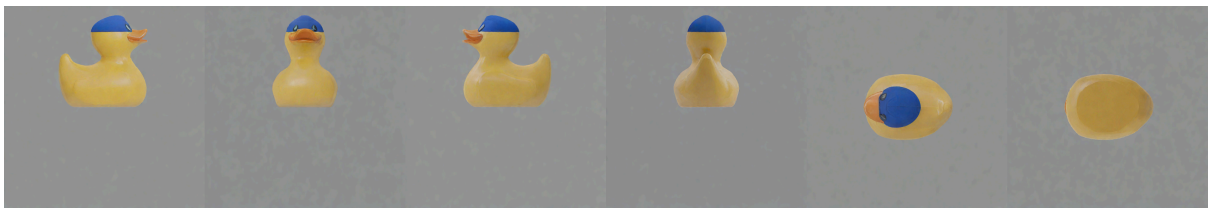


Figure 15: Inference avec prompt chantier gen1

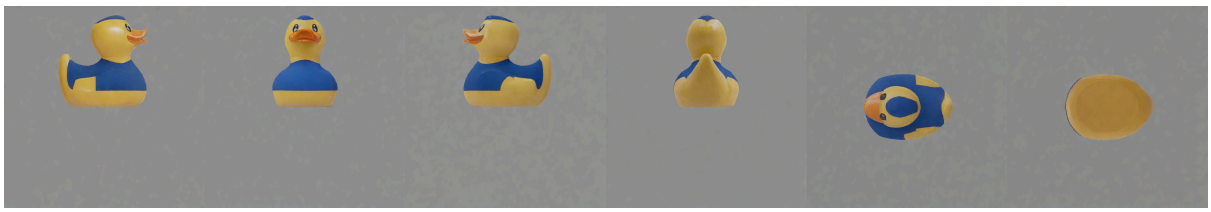


Figure 16: Inference avec prompt chantier gen1

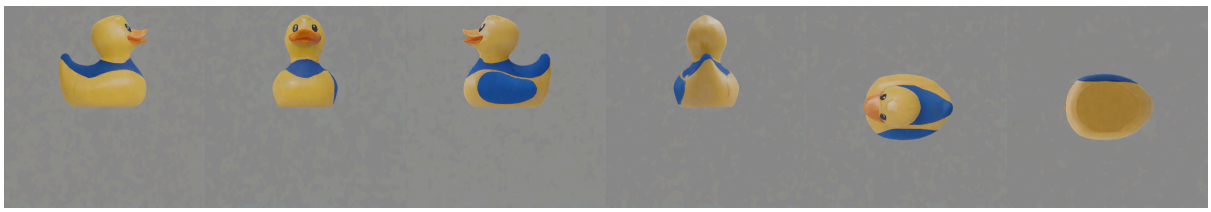


Figure 17: Inference avec prompt chantier gen1

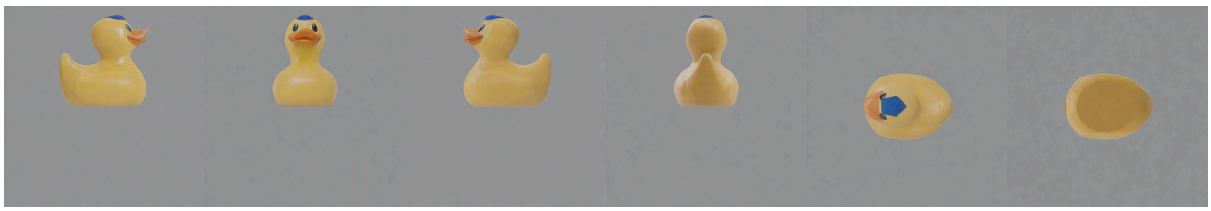


Figure 18: Inference avec prompt chantier gen1

Le canard pompier:

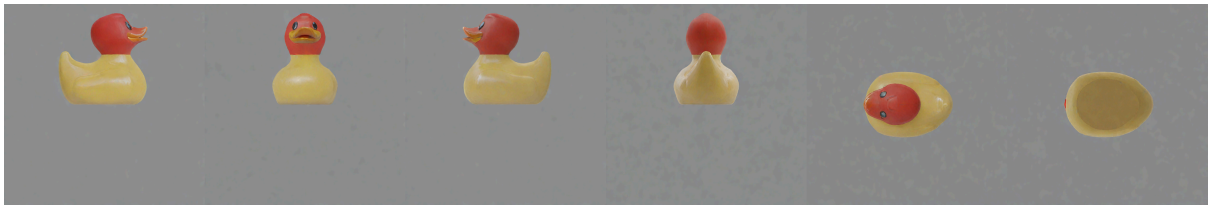


Figure 19: Inference avec prompt pompier gen1

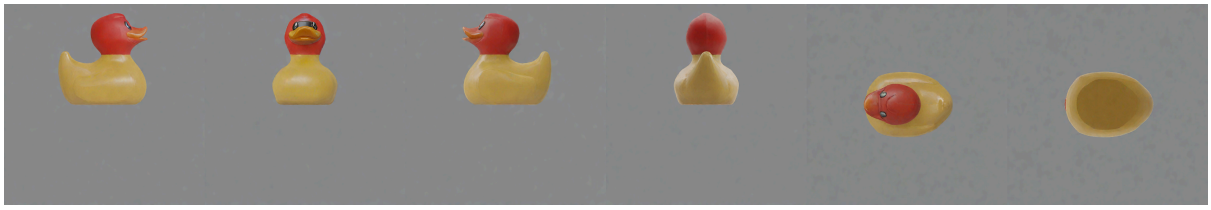


Figure 20: Inference avec prompt pompier gen1

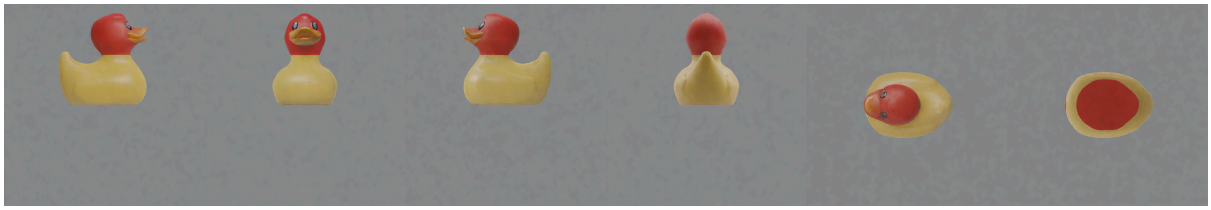


Figure 21: Inference avec prompt pompier gen1

Figure 22: Inference avec prompt pompier gen1

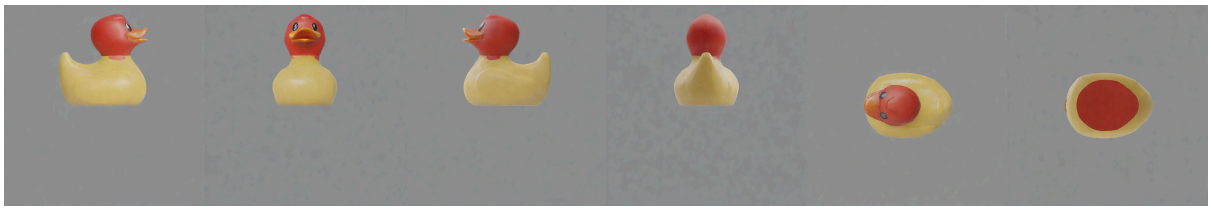


Figure 23: Inference avec prompt pompier gen1

Le canard astronaute:

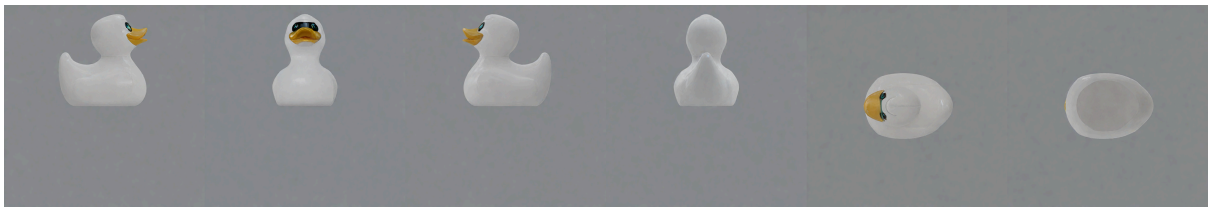


Figure 24: Inference avec prompt astronaute gen1

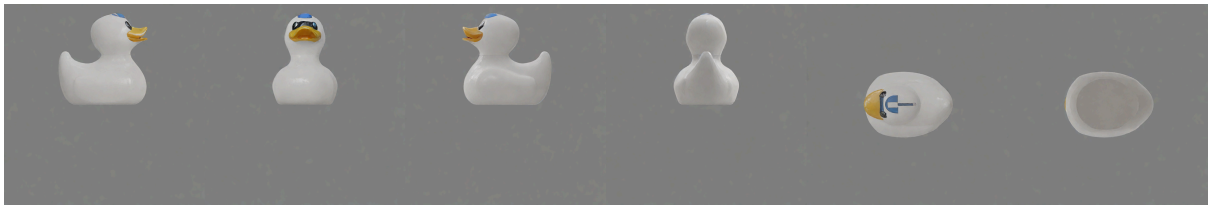


Figure 25: Inference avec prompt astronaute gen1

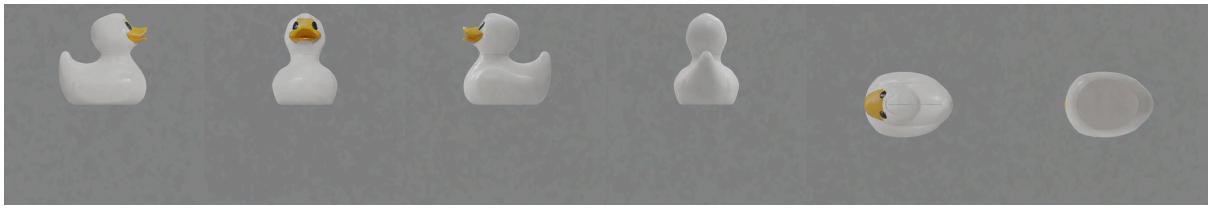


Figure 26: Inference avec prompt astronaute gen1

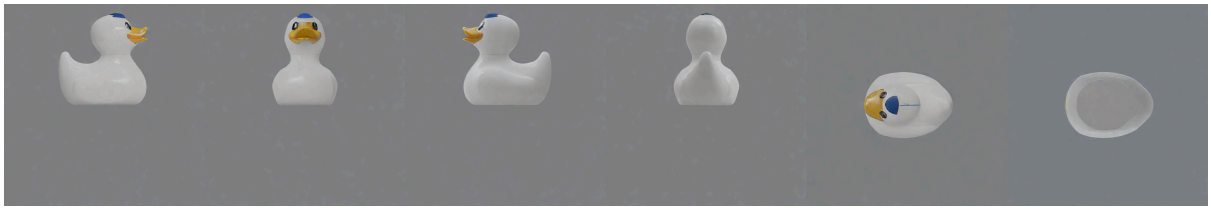


Figure 27: Inference avec prompt astronaute gen1

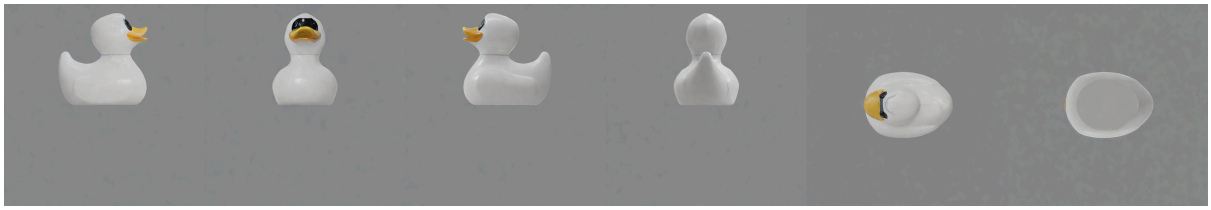


Figure 28: Inference avec prompt astronaute gen1

1.5. Evaluation

L'évaluation humaine est de constater que les traits qui concernent un pompier, astronaute ou du chantier n'apparaissent plus.

Il a cependant très bien appris à placer les couleurs au niveau de la tête.

1.6. Finetuning data augmented

Le dataset est maintenant composé de 200 canards de textures différentes.

Les canards ont été dupliqué par 10. Les données de couleur ont été légerment haltéré.

Le but est maintenant d'entrainer le modèle pré-entrainé sur les multiview des canards présentés.

1.6.1. Inference

Ici l'inférence avec nos poids custom issue de l'entraînement.

Le canard chantier:

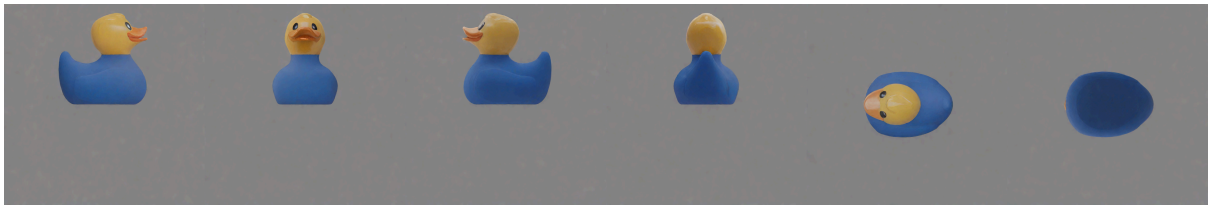


Figure 29: Inference avec prompt chantier gen1

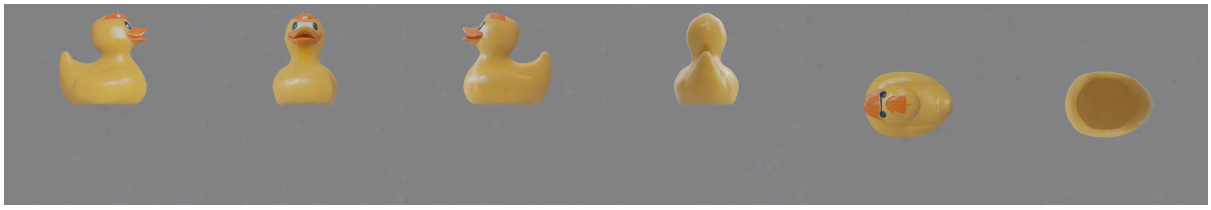


Figure 30: Inference avec prompt chantier gen2

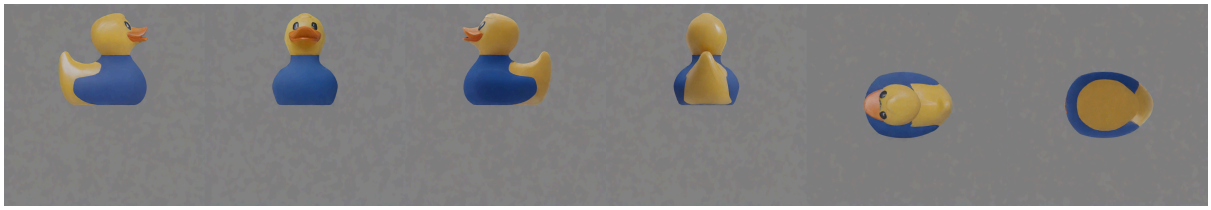


Figure 31: Inference avec prompt chantier gen3

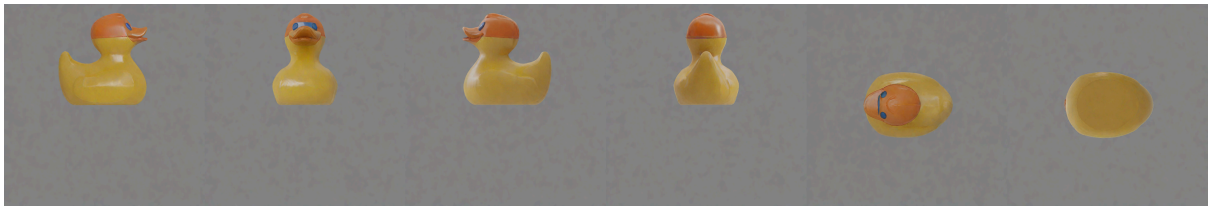


Figure 32: Inference avec prompt chantier

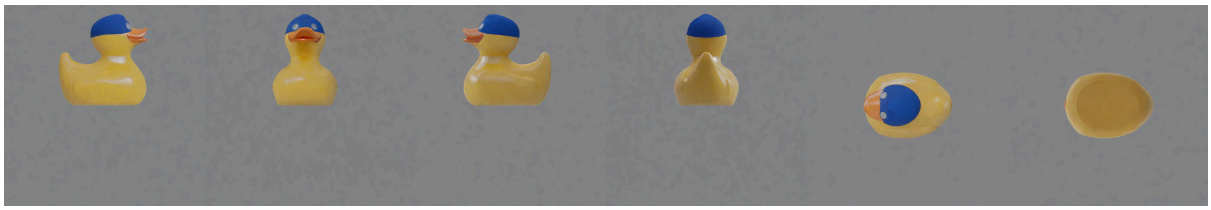


Figure 33: Inference avec prompt chantier

Le canard pompier

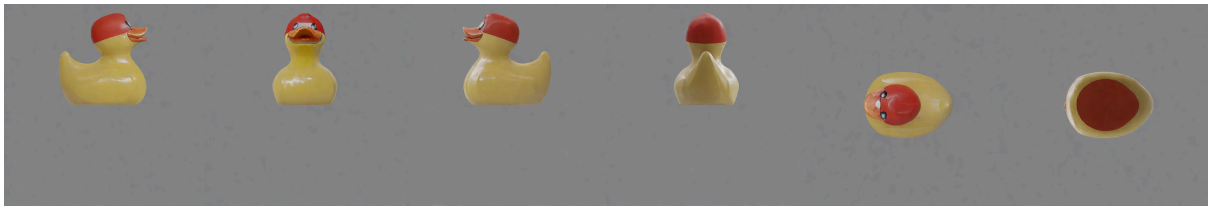


Figure 34: Inference avec prompt pompier

Figure 35: Inference avec prompt pompier

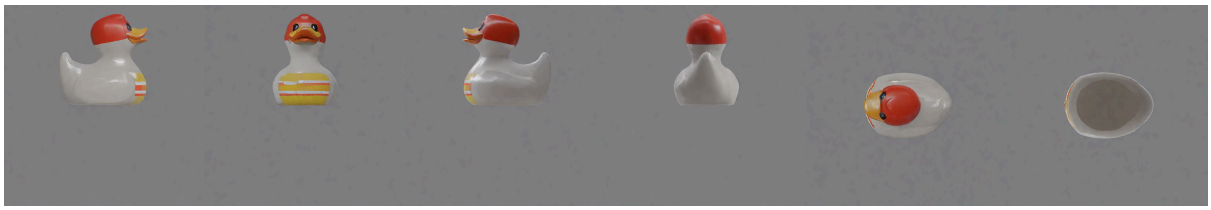


Figure 36: Inference avec prompt pompier

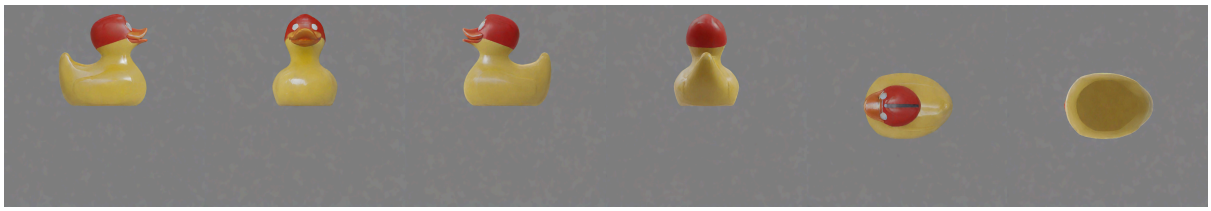


Figure 37: Inference avec prompt pompier

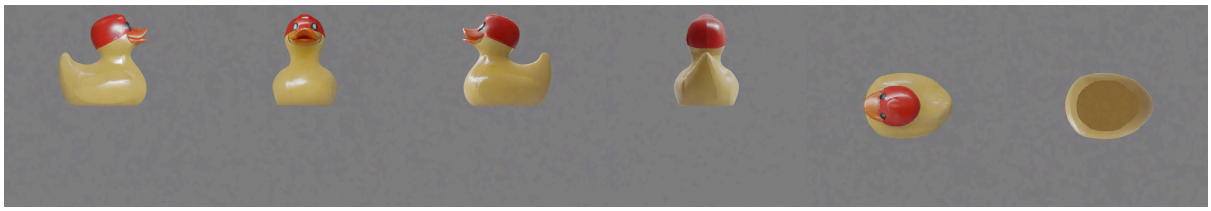


Figure 38: Inference avec prompt pompier

Le canard astronaute:

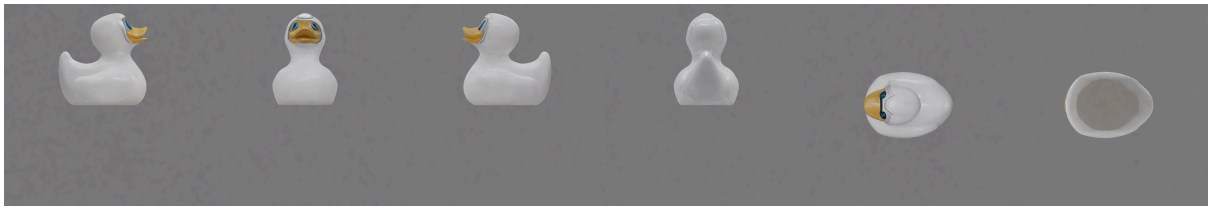


Figure 39: Inference avec prompt astronaute

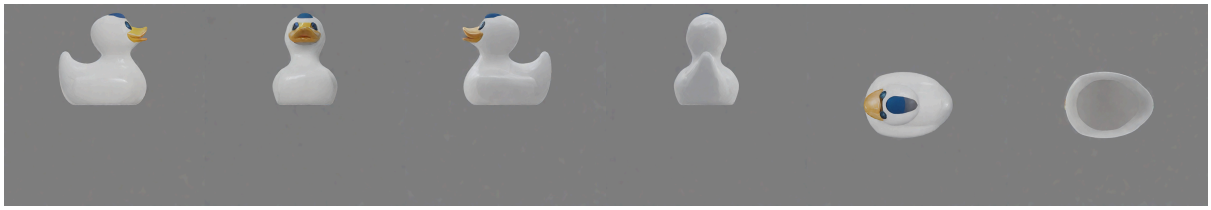


Figure 40: Inference avec prompt astronaute

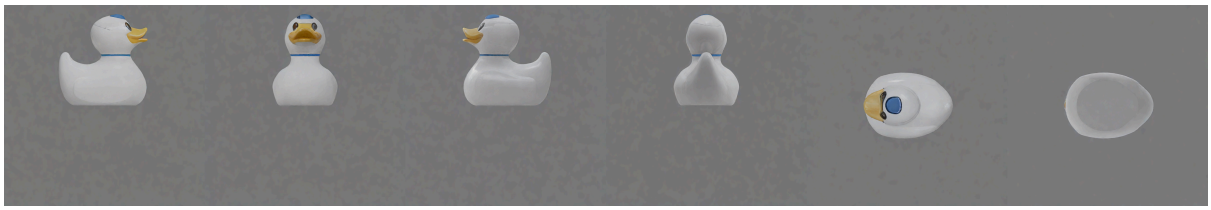


Figure 41: Inference avec prompt astronaute

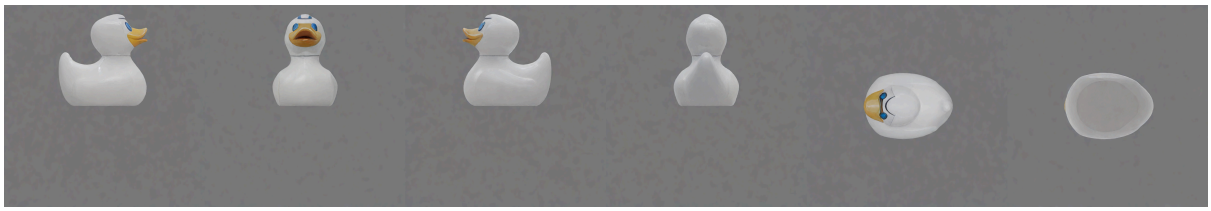


Figure 42: Inference avec prompt astronaute

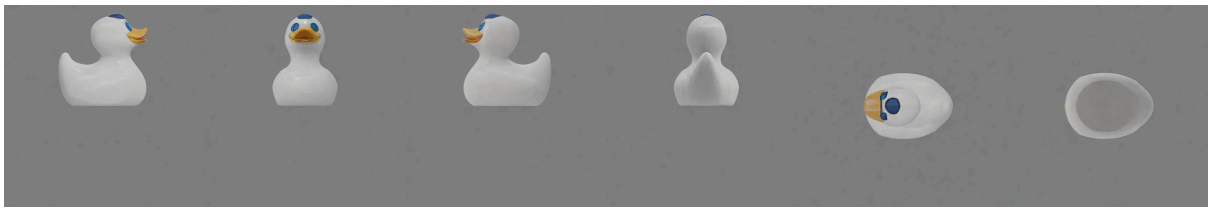


Figure 43: Inference avec prompt astronaute

1.7. Evaluation

Précédent finetuning: L'évaluation humaine est de constater que les traits qui concernent un pompier, astronaute ou du chantier n'apparaissent plus.

Il a cependant très bien appris à placer les couleurs au niveau de la tête.

Ici, il n'y pas d'amélioration lors de l'inférence après l'augmentation de donnée.

2. Décision Finetuning

L'inférence n'a pas donné les résultats attendues.

L'apprentissage suite au finetuning réduit la qualité des canards générés. En effet, le prompt est moins bien respecté.

2.1. Phase de production

Plus de temps doit être accordé au finetuning, afin de pouvoir remplacer le modèle pré-entrainé par un modèle custom. Un dataset plus grand et plus varié doit être généré. La génération de données (mesh puis multiview puis training) peut être facilement étendue. Les scripts SLURM permettent d'automatiser les runs

2.2. Apprentissage

Le finetuning dépend fortement de ce que le modèle sait déjà faire. Peu de données, le modèle apprend surtout des patterns simples comme des couleurs, mais pas des concepts complexes.