🤖 ArBot - Guide Pédagogique Complet

Runbook Séquencé et Opérationnel

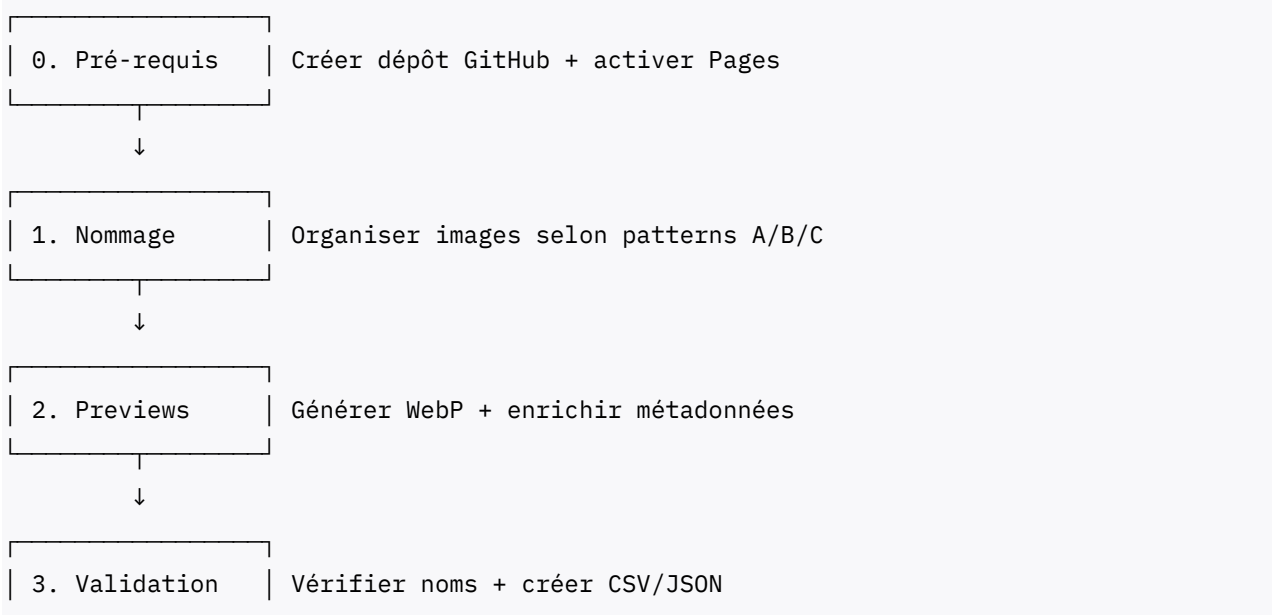
📋 Vue d'ensemble du projet

ArBot est un système de gestion et d'analyse d'images techniques pour la documentation de chantiers et sinistres. Le projet utilise GitHub Pages pour héberger publiquement les images, documents et métadonnées, permettant une analyse automatisée via des scripts Python et l'intelligence artificielle[1].

Architecture globale:

- 🗂️ **Stockage:** GitHub (dépôt public)
- 🌐 **Publication:** GitHub Pages
- 🐍 **Automatisation:** Scripts Python
- 🤖 **Analyse:** Vision GPT-4o

📊 Diagramme du flux de travail



```
└──────────┬──────────┘
           ↓
┌──────────┴──────────┐
│ 4. Index docs       │ Cataloguer DTU et notices PDF
└──────────┬──────────┘
           ↓
┌──────────┴──────────┐
│ 5. Galerie HTML     │ Page web de visualisation
└──────────┬──────────┘
           ↓
┌──────────┴──────────┐
│ 6-8. Options        │ Crops, Tuilage, Checksums
└──────────┬──────────┘
           ↓
┌──────────┴──────────┐
│ 9. Publication      │ Push vers GitHub
└──────────┬──────────┘
           ↓
┌──────────┴──────────┐
│ 10-12. Analyse      │ Automatisation + Vision IA
└─────────────────────┘
```

🎯 Étape 0 : Pré-requis

Objectif

Préparer l'environnement de travail avec GitHub et Python pour héberger publiquement vos images et documents[1].

Actions détaillées

1. Créer le dépôt GitHub

- Aller sur github.com et se connecter
- Cliquer sur "New repository"
- Nom du dépôt: `Arbot-MiniDB` (respecter exactement la casse)

- Cocher "Public" (obligatoire pour les URLs publiques)
- Créer le dépôt

2. Activer GitHub Pages

- Dans le dépôt, aller dans `Settings` → `Pages`
- Source: sélectionner `main` (branche)
- Folder: sélectionner `/ (root)`
- Sauvegarder

3. Installer Python

- Télécharger Python 3.11+ sur python.org
- Installer en cochant "Add to PATH"
- Vérifier: `python --version` dans le terminal

Structure de dossiers à créer

```
Arbot-MiniDB/
├── images/          # Images originales
├── docs/            # Documents PDF (DTU, notices)
├── json/            # Fichiers de métadonnées
└── scripts/         # Scripts Python
```

✅ Points de vérification

- [ ] Dépôt public créé
- [ ] GitHub Pages activé
- [ ] Python 3.11+ installé
- [ ] Dossiers de base créés

⚠ Pièges à éviter

- **Dépôt privé:** Les URLs RAW ne fonctionneront pas
- **Casse incorrecte:** `arbot-minidb` ≠ `Arbot-MiniDB`
- **Données GPS:** Nettoyer les métadonnées EXIF avant publication

---

🏷️ Étape 1 : Structuration et nommage des images

Objectif

Organiser les images selon une convention de nommage stricte permettant l'analyse automatique par expression régulière (regex)[1].

Système de nommage : 3 cas

Cas A : Chantier (id_zone = 0)

```
Pattern: XXXX_DETAIL_SPEC_AAAAMMJJ.ext
Exemple: 0123_DETAIL_facade_20241015.jpg
```

- `0123` = Id_File (identifiant unique)
- `facade` = Spécification technique
- `20241015` = Date (15 octobre 2024)
- **Classificaton:** "chantier" car date présente

Cas B : Plan (id_zone = 1)

```
Pattern: XXXX_DETAIL_YYYY.ext
Exemple: 1245_DETAIL_0012.jpg
```

- `1245` = Id_File
- `0012` = Numéro de référence plan

Cas C : Sinistre (id_zone ∈ {2..9})

```
Pattern: XXXX_DETAIL_SPEC.ext
Exemple: 2678_DETAIL_humidite.jpg
```

- `2678` = Id_File
- `humidite` = Spécification du sinistre

- **Classification:** "sinistre" (pas de date)

Formule de calcul Id_File

$$Id\_File = Id\_Zone \times 1000 + Id\_Cat \times 100 + id\_image$$

Exemple pratique:

- Id_Zone = 2 (sinistre)
- Id_Cat = 3 (catégorie humidité)
- id_image = 45 (45ème image de cette catégorie)
- **Id_File = 2×1000 + 3×100 + 45 = 2345**

Organisation des fichiers

```
images/
├── chantier/
│   ├── 0123_DETAIL_facade_20241015.jpg
│   └── 0124_DETAIL_toiture_20241016.jpg
├── plans/
│   ├── 1245_DETAIL_0012.jpg
│   └── 1246_DETAIL_0013.jpg
└── sinistre/
    ├── 2678_DETAIL_humidite.jpg
    └── 2679_DETAIL_fissure.jpg
```

✅ Points de vérification

- [ ] Noms conformes aux patterns A/B/C
- [ ] Id_File unique pour chaque image
- [ ] Dates au format AAAAMMJJ (Cas A)
- [ ] Spécifications listées dans le dictionnaire

⚠ Erreurs fréquentes

- **Espaces:** `DETAIL facade` → **NON** | `DETAIL_facade` → **OUI**
- **Extensions:** Utiliser `.jpg`, `.png` (minuscules)

- **Dates manquantes:** Cas A doit avoir AAAAMMJJ
- **Id_File incohérent:** Vérifier la formule de calcul

🖼️ Étape 2 : Génération de previews WebP

Objectif

Créer des versions allégées des images pour accélérer le chargement web et l'analyse IA, tout en corrigeant automatiquement l'orientation EXIF[1].

Pourquoi WebP ?

- **Compression:** 25-35% plus léger que JPEG
- **Qualité:** Conservation visuelle excellente
- **Performance:** Chargement rapide pour analyse IA

Installation des dépendances

```
pip install pillow
```

Script à créer : `make_previews_and_augment_json.py`

Placer ce fichier à la racine du projet[1].

Commande d'exécution

```
python make_previews_and_augment_json.py ^
  --images .\images ^
  --out_json .\json\images_db.json ^
  --owner Tazevil --repo Arbot-MiniDB --branch main
```

Paramètres expliqués

Paramètre	Description	Exemple
`--images`	Dossier source des images	`.\images`

`--out_json`	Fichier JSON de sortie	`.\json\images_db.json`
`--owner`	Propriétaire GitHub	`Tazevil`
`--repo`	Nom du dépôt	`Arbot-MiniDB`
`--branch`	Branche	`main`

Résultat de l'exécution

Fichiers créés:

```
images/
└── preview/
    ├── 0123_DETAIL_facade_20241015.webp
    ├── 0124_DETAIL_toiture_20241016.webp
    └── ...
```

JSON enrichi (`json/images_db.json`):

```
{
  "0123_DETAIL_facade_20241015.jpg": {
    "id_file": "0123",
    "detail": "facade",
    "date": "20241015",
    "url_original": "https://raw.githubusercontent.com/.../images/...",
    "url_preview": "https://raw.githubusercontent.com/.../preview/...webp",
    "roi_hints": {"x": 0.5, "y": 0.5, "w": 0.3, "h": 0.3},
    "phase": "chantier"
  }
}
```

Dimensions des previews

- **Côté long cible:** 1280-2048 pixels
- **Ratio:** Conservé (pas de déformation)
- **Orientation:** Corrigée automatiquement selon EXIF

✅ Points de vérification

- [ ] Dossier `images/preview/` créé
- [ ] Fichiers WebP générés
- [ ] `images_db.json` contient `url_preview`
- [ ] Champs `roi_hints` et `phase` ajoutés

⚠ Erreurs fréquentes

- **Pillow manquant:** Installer avec `pip install pillow`
- **Previews lourds:** Vérifier dimensions (max 2048px)
- **Orientation incorrecte:** Le script corrige automatiquement EXIF

✔️ Étape 3 : Validation et indexation

Objectif

Valider la conformité des noms de fichiers, vérifier l'unicité des Id_File, et générer deux fichiers d'index : CSV et JSON[1].

Script à utiliser : `validate_pack.py`

Version patchée avec support récursif (`rglob`) pour sous-dossiers[1].

Commande d'exécution

```
python validate_pack.py --images .\images --out .\json ^
  --owner Tazevil --repo Arbot-MiniDB --branch main
```

Validations effectuées

1. Conformité regex (Patterns A/B/C)

```
✅ VALIDE: 0123_DETAIL_facade_20241015.jpg
X INVALIDE: 0123_DETAILfacade_20241015.jpg (underscore manquant)
X INVALIDE: 123_DETAIL_facade.jpg (Id_File trop court)
```

2. Unicité Id_File

```
✅ UNIQUE: Chaque Id_File apparaît une seule fois
X DUPLICAT: 2345 apparaît 2 fois → ERREUR
```

3. Cohérence Id_File

```
Vérification: Id_File = Id_Zone × 1000 + Id_Cat × 100 + id_image
✅ COHÉRENT: 2345 pour Zone=2, Cat=3, Image=45
X INCOHÉRENT: 2350 pour Zone=2, Cat=3, Image=45
```

Fichiers générés

1. `json/images_sidecar.csv`

```
filename,id_file,id_zone,id_cat,id_image,detail,spec,date,phase,status
0123_DETAIL_facade_20241015.jpg,0123,0,1,23,facade,,20241015,chantier,OK
2345_DETAIL_humidite.jpg,2345,2,3,45,humidite,humidite,,sinistre,OK
```

2. `json/images_db.json`

Fichier JSON structuré avec métadonnées complètes[1].

Tableau des statuts possibles

Statut	Signification	Action requise
`OK`	Fichier valide	Aucune
`DUPLICAT`	Id_File en double	Renommer
`INCOHERENT`	Formule Id_File incorrecte	Recalculer
`REGEX_FAIL`	Pattern non conforme	Corriger nom

✅ Points de vérification

- [ ] Aucune ligne avec statut `DUPLICAT`
- [ ] Aucune ligne avec statut `INCOHERENT`
- [ ] CSV lisible dans Excel/LibreOffice

- [ ] JSON valide (tester avec jsonlint.com)

⚠ Messages d'erreur courants

"fatal: not a git repository"

- **Cause:** Exécution hors d'un dépôt Git
- **Solution:** Ignorable si vous ne faites pas de `git push` local

"utcnow() deprecated"

- **Cause:** Méthode obsolète Python 3.12+
- **Solution:** Le script utilise `now(timezone.utc)`

🔄 Ordre d'exécution

Si vous avez déjà exécuté l'étape 2:

1. **Option A:** Valider puis enrichir (étape 3 → étape 2)
2. **Option B:** Enrichir puis valider (étape 2 → étape 3)
3. **Recommandé:** Valider d'abord pour détecter les erreurs

📚 Étape 4 : Index des documents techniques

Objectif

Créer un catalogue JSON des documents PDF (DTU, notices techniques) pour permettre les citations précises avec `doc_id + page` lors de l'analyse IA[1].

Qu'est-ce qu'un DTU ?

DTU = Documents Techniques Unifiés

- Normes françaises du bâtiment
- Règles de l'art pour la construction
- Exemple: DTU 20.1 (Maçonnerie), DTU 40.11 (Couverture)

Organisation des documents

```
docs/
├── DTU_20.1_Maconnerie.pdf
├── DTU_40.11_Couverture.pdf
├── Notice_Etancheite_2023.pdf
└── Guide_Diagnostic_Humidite.pdf
```

Script à créer : `scripts/generate_docs_index.py`

Commande d'exécution

```
python .\scripts\generate_docs_index.py
```

Structure du fichier généré : `json/docs_index.json`

```
{
  "documents": [
    {
      "id": "DTU_20.1",
      "title": "DTU 20.1 - Maçonnerie des ouvrages en béton",
      "type": "DTU",
      "url":
"https://raw.githubusercontent.com/Tazevil/Arbot-MiniDB/main/docs/DTU_20.1_Maconnerie.pd
f"
    },
    {
      "id": "NOTICE_ETANCH",
      "title": "Notice technique - Étanchéité 2023",
      "type": "notice",
      "url":
"https://raw.githubusercontent.com/Tazevil/Arbot-MiniDB/main/docs/Notice_Etancheite_2023
.pdf"
    }
  ],
  "last_updated": "2024-11-07T12:30:00Z"
}
```

Champs expliqués

Champ	Description	Exemple
`id`	Identifiant court et stable	`DTU_20.1`
`title`	Titre complet et explicite	`DTU 20.1 - Maçonnerie`
`type`	Catégorie du document	`DTU`, `notice`, `guide`
`url`	URL RAW GitHub publique	`https://raw.githubusercontent.com/...`

Utilisation dans l'analyse IA

Citation avec page:

```
{
  "anomaly": "Fissure structurelle",
  "reference": {
    "doc_id": "DTU_20.1",
    "pages": [15, 16],
    "excerpt": "Les fissures traversantes..."
  }
}
```

✅ Points de vérification

- [ ] Tous les PDF placés dans `docs/`
- [ ] URLs RAW fonctionnelles (tester dans navigateur)
- [ ] `doc_id` courts et sans espaces
- [ ] Titres explicites et complets

⚠️ Limitations importantes

PDF scannés vs PDF natifs:

- **PDF natif:** Texte extractible → Citations précises ✅
- **PDF scanné:** Images → Citations pages en `UNKNOWN` ❌

- **Solution:** Utiliser OCR (Tesseract) ou refaire PDF

Accessibilité:

- URLs privées ne fonctionnent pas
- Vérifier que le dépôt est bien public

🌐 Étape 5 : Galerie HTML publique

Objectif

Générer une page web interactive affichant toutes les images pour le contrôle visuel rapide du projet[1].

Script à créer : `generate_index_html.py`

Script récursif parcourant tous les sous-dossiers d'images[1].

Commande d'exécution

```
python generate_index_html.py --images .\images --out .\index.html
```

Structure HTML générée

```
<!DOCTYPE html>
<html>
<head>
    <title>ArBot - Galerie d'images</title>
    <style>
        .gallery { display: grid; grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); gap: 20px; }
        .card { border: 1px solid #ddd; padding: 10px; border-radius: 8px; }
        img { width: 100%; height: auto; }
    </style>
</head>
<body>
    <h1>🤖 ArBot - Galerie complète</h1>
    <div class="gallery">
```

```
        <div class="card">
            <img src="images/chantier/0123_DETAIL_facade_20241015.jpg" alt="facade">
            <p><strong>0123</strong> - facade (chantier)</p>
            <p>Date: 2024-10-15</p>
        </div>
        <!-- ... autres images ... -->
    </div>
</body>
</html>
```

Fonctionnalités de la galerie

1. Organisation visuelle

- Grille responsive (s'adapte à la largeur d'écran)
- Cartes avec informations clés
- Aperçu des images

2. Filtrage (optionnel)

- Par phase (chantier/sinistre)
- Par zone (0-9)
- Par date

3. Navigation

- Liens vers images originales
- Liens vers previews WebP
- Métadonnées affichées

URL de publication

Une fois publié sur GitHub:

```
https://tazevil.github.io/Arbot-MiniDB/
```

✅ Points de vérification

- [ ] `index.html` créé à la racine
- [ ] Chemins relatifs utilisés (`images/...` pas `/images/...`)
- [ ] GitHub Pages activé (Settings → Pages)
- [ ] Page accessible via URL publique

⚠️ Erreurs fréquentes

Chemins absolus:

```
X <img src="/images/..." >  → Ne fonctionne pas sur GitHub Pages
✅ <img src="images/..." >   → Fonctionne correctement
```

Casse incorrecte:

```
X https://tazevil.github.io/arbot-minidb/
✅ https://tazevil.github.io/Arbot-MiniDB/
```

Délai de publication:

- Première activation: 1-10 minutes
- Mises à jour: cache CDN 1-5 minutes

🔍 Étape 6 : Crops depuis ROI (optionnel)

Objectif

Extraire des zones d'intérêt (ROI = Region of Interest) pour créer des vues rapprochées sur les défauts, améliorant la précision de l'analyse IA[1].

Qu'est-ce qu'un ROI ?

ROI = Rectangle de coordonnées définissant une zone précise:

```
roi_hints: {
  "x": 0.5,    // Position X du centre (0.0 à 1.0)
  "y": 0.5,    // Position Y du centre (0.0 à 1.0)
```

```
  "w": 0.3,    // Largeur (30% de l'image)
  "h": 0.3     // Hauteur (30% de l'image)
}
```

Exemple visuel

```
┌─────────────────────────────────┐
│                                 │
│         Image complète          │
│                                 │
│     ┌──────────────┐            │  ← ROI centré
│     │   DÉFAUT     │            │
│     │   (crop)     │            │
│     └──────────────┘            │
│                                 │
└─────────────────────────────────┘
```

Script à utiliser : `make_crops_from_roi.py`

Commande d'exécution

```
python make_crops_from_roi.py
```

Fichiers générés

```
images/
├── crop/
│   ├── 0123_DETAIL_facade_20241015_crop.jpg
│   ├── 2345_DETAIL_humidite_crop.jpg
│   └── ...
```

Mise à jour JSON

Avant:

```
{
  "url_original": "https://.../0123_DETAIL_facade_20241015.jpg",
```

```
  "url_preview": "https://.../preview/0123_DETAIL_facade_20241015.webp",
  "roi_hints": {"x": 0.5, "y": 0.5, "w": 0.3, "h": 0.3}
}
```

Après:

```
{
  "url_original": "https://.../0123_DETAIL_facade_20241015.jpg",
  "url_preview": "https://.../preview/0123_DETAIL_facade_20241015.webp",
  "url_crop": "https://.../crop/0123_DETAIL_facade_20241015_crop.jpg",
  "roi_hints": {"x": 0.5, "y": 0.5, "w": 0.3, "h": 0.3}
}
```

Cas d'usage

Quand utiliser les crops ?

- ✅ Défauts localisés (fissure, tache)
- ✅ Détails techniques précis
- ✅ Analyse fine de texture

Quand utiliser l'image complète ?

- ✅ Vue d'ensemble
- ✅ Contexte spatial
- ✅ Plusieurs zones d'intérêt

Ajuster les ROI manuellement

Éditer `json/images_db.json`:

```
{
  "2345_DETAIL_humidite.jpg": {
    "roi_hints": {
      "x": 0.3,    // Décalé vers la gauche
      "y": 0.7,    // Décalé vers le bas
      "w": 0.2,    // Plus petit (20%)
      "h": 0.2
```

```
    }
  }
}
```

Puis relancer:

```
python make_crops_from_roi.py
```

✅ Points de vérification

- [ ] Dossier `images/crop/` créé
- [ ] Crops générés avec suffix `_crop.jpg`
- [ ] `images_db.json` contient `url_crop`
- [ ] URLs RAW fonctionnelles

⚠️ Erreurs fréquentes

ROI hors limites:

```
# Si x=0.9 et w=0.3 → dépasse l'image (0.9+0.15=1.05 > 1.0)
# Le script ajuste automatiquement
```

Absence de roi_hints:

- Le script utilise un ROI centré par défaut
- Taille: 30% × 30% de l'image

🧩 Étape 7 : Tuilage 1024px (optionnel)

Objectif

Découper les grandes images en tuiles (patches) de 1024×1024 pixels avec recouvrement pour détecter les petits défauts invisibles sur les previews[1].

Pourquoi le tuilage ?

Problème: Grande image 6000×4000px

- Preview 2048px → Perte de détails fins
- IA peut manquer petites fissures

Solution: Tuiles 1024×1024px + 10% recouvrement

- Chaque zone analysée en haute résolution
- Recouvrement évite les défauts sur les bords

Visualisation du tuilage

```
┌─────────────────────────────────────┐
│  [Tuile 1]  │ [Tuile 2]  │ [Tuile 3]│
│             │            │          │
├─────────────┼────────────┼──────────┤
│  [Tuile 4]  │ [Tuile 5]  │ [Tuile 6]│  ← Recouvrement 10%
│             │            │          │
├─────────────┼────────────┼──────────┤
│  [Tuile 7]  │ [Tuile 8]  │ [Tuile 9]│
│             │            │          │
└─────────────────────────────────────┘

    Image originale 3000×2000px
```

Script à utiliser : `tile_1024_overlap.py`

Commande d'exécution

```
python tile_1024_overlap.py --src .\images --out .\tiles --size 1024 --overlap 0.10
```

Paramètres expliqués

Paramètre	Valeur	Description
`--src`	`.\images`	Dossier source
`--out`	`.\tiles`	Dossier destination
`--size`	`1024`	Taille des tuiles (px)

`--overlap`	`0.10`	Recouvrement 10%

Fichiers générés

Structure:

```
tiles/
├── 0123_DETAIL_facade_20241015_tile_00.jpg
├── 0123_DETAIL_facade_20241015_tile_01.jpg
├── 0123_DETAIL_facade_20241015_tile_02.jpg
├── ...
└── tiles_map.json
```

Fichier de mapping (`tiles/tiles_map.json`):

```
{
  "0123_DETAIL_facade_20241015.jpg": {
    "original_size": [4000, 3000],
    "tile_size": 1024,
    "overlap": 0.10,
    "tiles": [
      {
        "filename": "0123_DETAIL_facade_20241015_tile_00.jpg",
        "position": {"x": 0, "y": 0},
        "size": [1024, 1024]
      },
      {
        "filename": "0123_DETAIL_facade_20241015_tile_01.jpg",
        "position": {"x": 922, "y": 0},
        "size": [1024, 1024]
      }
    ]
  }
}
```

Calcul du nombre de tuiles

Formule:

$$Nb\ tuiles = \lceil \frac{largeur}{size \times (1-overlap)} \rceil \times \lceil \frac{hauteur}{size \times (1-overlap)} \rceil$$

Exemple: Image 4000×3000px, tuile 1024px, overlap 10%

- Step effectif: $1024 \times 0.9 = 922$ px
- Tuiles horizontales: $\lceil 4000/922 \rceil = 5$
- Tuiles verticales: $\lceil 3000/922 \rceil = 4$
- **Total: 5 × 4 = 20 tuiles**

Stratégie d'analyse

1. Analyse en cascade:

```
Preview (rapide) → Crops (zones suspectes) → Tuiles (confirmation)
```

2. Priorisation:

- Commencer par previews
- Tuiler uniquement images critiques
- Éviter surcharge API

✅ Points de vérification

- [ ] Dossier `tiles/` créé
- [ ] Tuiles générées avec suffix `_tile_XX.jpg`
- [ ] `tiles_map.json` créé
- [ ] Position et taille cohérentes

⚠ Considérations importantes

Charge API:

- 20 tuiles × $0.02/appel = 0.40$ par image
- Prioriser les images vraiment nécessaires

Stockage:

- 1 image 4000×3000px = 5 MB

- 20 tuiles = 20 MB total
- Prévoir espace GitHub suffisant

🔐 Étape 8 : Checksums SHA-256 (traçabilité)

Objectif

Générer des empreintes cryptographiques (hash SHA-256) de tous les fichiers pour garantir l'intégrité et la traçabilité du projet[1].

Qu'est-ce qu'un hash SHA-256 ?

Hash = Empreinte numérique unique d'un fichier

- Longueur fixe: 64 caractères hexadécimaux
- Modification du fichier → Hash différent
- Collision quasi-impossible

Exemple:

```
Fichier: 0123_DETAIL_facade_20241015.jpg
SHA-256: a3f5c8d9e2b1f7a6c4d8e9f2b3a5c7d9e1f3a5b7c9d2e4f6a8b1c3d5e7f9a2b4
```

Outils et scripts

1. Script Batch Windows (`scripts/generate_checksum.bat`):

```
@echo off
certutil -hashfile images\*.jpg SHA256 > checksums_sha256.txt
certutil -hashfile json\*.json SHA256 >> checksums_sha256.txt
```

2. Script Python (`build_manifest.py`):
Génère un manifest JSON avec hash global du projet[1].

Commandes d'exécution

Windows:

```
scripts\generate_checksum.bat
```

Python (tous OS):

```
python build_manifest.py
```

Fichiers générés

1. `checksums_sha256.txt` (format texte):

```
SHA256(images/chantier/0123_DETAIL_facade_20241015.jpg) = a3f5c8d9...
SHA256(images/sinistre/2345_DETAIL_humidite.jpg) = b7d2e4f6...
SHA256(json/images_db.json) = c9e1f3a5...
```

2. `manifest.json` (format structuré):

```
{
  "project": "ArBot-MiniDB",
  "version": "1.0.0",
  "generated_at": "2024-11-07T12:30:00Z",
  "files": {
    "images/chantier/0123_DETAIL_facade_20241015.jpg": {
      "sha256": "a3f5c8d9e2b1f7a6c4d8e9f2b3a5c7d9e1f3a5b7c9d2e4f6a8b1c3d5e7f9a2b4",
      "size": 2548736,
      "last_modified": "2024-10-15T14:23:00Z"
    }
  },
  "global_sha256": "e7f9a2b4c6d8e1f3a5b7c9d2e4f6a8b1c3d5e7f9a2b4c6d8e1f3a5b7c9d2e4f6"
}
```

Utilisation du hash global

Vérification d'intégrité:

```
# Télécharger le projet
git clone https://github.com/Tazevil/Arbot-MiniDB.git
cd Arbot-MiniDB
```

```
# Régénérer le hash
python build_manifest.py

# Comparer avec hash original
# Si identique → Aucune modification
# Si différent → Fichiers modifiés ou corrompus
```

Exclusions automatiques

Le script `build_manifest.py` exclut:

- `.git/` (dépôt Git)
- `.github/` (workflows)
- `node_modules/`
- Fichiers temporaires (`.tmp`, `.bak`)

✅ Points de vérification

- [ ] `checksums_sha256.txt` généré
- [ ] `manifest.json` créé
- [ ] Champ `global_sha256` présent
- [ ] Hash stable entre 2 clones identiques

⚠️ Quand générer les checksums ?

Bon moment:

- ✅ Après ajout final des images
- ✅ Avant publication sur GitHub
- ✅ À chaque version stable

Mauvais moment:

- ❌ Pendant modifications actives
- ❌ Avant validation complète

🚀 Étape 9 : Publication sur GitHub

Objectif

Publier tous les fichiers sur GitHub pour les rendre accessibles publiquement via URLs RAW et GitHub Pages[1].

Trois méthodes de publication

Méthode 1 : Interface Web GitHub (Débutants)

Étapes:

1. Aller sur `github.com/Tazevil/Arbot-MiniDB`
2. Cliquer sur `Add file` → `Upload files`
3. Glisser-déposer les dossiers:
   - `images/`
   - `json/`
   - `docs/`
   - `index.html`
4. Ajouter message de commit: `"feat: ajout images et index"`
5. Cliquer sur `Commit changes`

Avantages:

- ✅ Pas besoin de Git
- ✅ Interface visuelle
- ✅ Rapide pour petits projets

Inconvénients:

- ❌ Limite 100 fichiers par upload
- ❌ Pas de gestion de versions avancée

Méthode 2 : GitHub Codespaces (Intermédiaire)

Étapes:

1. Sur GitHub, cliquer sur `Code` → `Codespaces` → `Create codespace on main`
2. Dans le terminal Codespaces:

```
# Copier fichiers locaux vers Codespace (drag & drop dans sidebar)
# Ou créer/éditer directement

# Générer previews et index
python make_previews_and_augment_json.py ...
python validate_pack.py ...
python generate_index_html.py ...

# Commit et push
git add .
git commit -m "feat: previews, indexes, page"
git push
```

Avantages:

- ✅ Environnement Python préconfiguré
- ✅ Exécution des scripts en ligne
- ✅ Pas d'installation locale

Méthode 3 : Git Local (Avancé)

Prérequis:

- Git installé localement
- Compte GitHub configuré

Commandes complètes:

```
# Initialiser le dépôt local
git init
```

```
# Ajouter tous les fichiers
git add .

# Premier commit
git commit -m "feat: previews, indexes, page"

# Renommer branche en main
git branch -M main

# Lier au dépôt distant
git remote add origin https://github.com/Tazevil/Arbot-MiniDB.git

# Pousser vers GitHub
git push -u origin main
```

Authentification HTTPS:

```
# Demander un Personal Access Token (PAT)
# Settings → Developer settings → Personal access tokens
# Utiliser le PAT comme mot de passe lors du push
```

Mises à jour ultérieures:

```
# Ajouter nouveaux fichiers
git add images/nouvelle_image.jpg

# Commit
git commit -m "add: nouvelle image facade"

# Push
git push
```

URLs générées après publication

1. URLs RAW (fichiers bruts):

```
https://raw.githubusercontent.com/Tazevil/Arbot-MiniDB/main/images/chantier/0123_DETAIL_facade_20241015.jpg
```

```
https://raw.githubusercontent.com/Tazevil/Arbot-MiniDB/main/json/images_db.json
```

2. URL GitHub Pages (HTML):

```
https://tazevil.github.io/Arbot-MiniDB/
```

Vérifications post-publication

Test URLs RAW:

```
# Copier URL d'une image dans navigateur
# Doit afficher l'image directement
```

Test GitHub Pages:

```
# Ouvrir https://tazevil.github.io/Arbot-MiniDB/
# Doit afficher la galerie HTML
```

✅ Points de vérification

- [ ] Tous les dossiers publiés (`images/`, `json/`, `docs/`)
- [ ] URLs RAW fonctionnelles
- [ ] GitHub Pages actif et accessible
- [ ] `index.html` s'affiche correctement

⚠️ Erreurs fréquentes

Casse des chemins:

```
X /Images/Chantier/... (majuscules)
✅ /images/chantier/... (minuscules)
```

Délai cache CDN:

- Première publication: 1-10 minutes
- Mises à jour: 1-5 minutes

- Forcer rafraîchissement: Ctrl+F5

Erreur 404:

- Vérifier que le dépôt est public
- Vérifier l'orthographe exacte des chemins
- Attendre quelques minutes (cache)

⚙️ Étape 10 : Automatisation GitHub Actions (optionnel)

Objectif

Automatiser la régénération des previews, validations, et index HTML à chaque ajout d'images via GitHub Actions[1].

Qu'est-ce que GitHub Actions ?

GitHub Actions = CI/CD (Continuous Integration/Deployment)

- Workflows automatiques déclenchés par événements
- Exécution dans machines virtuelles GitHub
- Gratuit pour dépôts publics

Cas d'usage ArBot

Déclencheurs:

- Ajout/modification d'images dans `images/**`
- Modification des scripts dans `scripts/`

Actions automatiques:

1. Générer previews WebP
2. Valider noms de fichiers
3. Créer index HTML
4. Commit automatique des résultats

Fichier workflow : .github/workflows/build-all.yml

Structure:

```
name: 🤖 ArBot - Génération automatique

on:
  push:
    paths:
      - 'images/**'
      - 'scripts/**'
  workflow_dispatch:  # Déclenchement manuel

jobs:
  build:
    runs-on: ubuntu-latest

    permissions:
      contents: write  # ⚠️ Obligatoire pour commit auto

    steps:
      - name: 📥 Checkout
        uses: actions/checkout@v4

      - name: 🐍 Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'

      - name: 📦 Install dependencies
        run: pip install pillow

      - name: 🖼️ Generate previews
        run: |
          python make_previews_and_augment_json.py \
            --images ./images \
            --out_json ./json/images_db.json \
            --owner Tazevil --repo Arbot-MiniDB --branch main

      - name: ✔️ Validate filenames
```

```
        run: |
          python validate_pack.py \
            --images ./images --out ./json \
            --owner Tazevil --repo Arbot-MiniDB --branch main

      - name: 🌐 Generate index.html
        run: |
          python generate_index_html.py \
            --images ./images --out ./index.html

      - name: 💾 Commit results
        run: |
          git config user.name "ArBot[bot]"
          git config user.email "arbot@users.noreply.github.com"
          git add images/preview/* json/* index.html
          git diff --quiet && git diff --staged --quiet || \
            git commit -m "🤖 auto: previews, validation, index"
          git push
```

Configuration étape par étape

1. Créer la structure:

```
mkdir -p .github/workflows
```

2. Créer le fichier:

```
# Dans .github/workflows/build-all.yml
# Copier le contenu YAML ci-dessus
```

3. Activer permissions:

- Sur GitHub: `Settings` → `Actions` → `General`
- `Workflow permissions` → Cocher `Read and write permissions`
- Sauvegarder

4. Tester le workflow:

```
# Méthode 1: Push une nouvelle image
git add images/nouvelle_image.jpg
git commit -m "add: nouvelle image"
git push

# Méthode 2: Déclenchement manuel
# GitHub → Actions → "ArBot - Génération automatique" → Run workflow
```

Visualisation du workflow

Onglet Actions sur GitHub:

```
🤖 ArBot - Génération automatique
└── build
    ├── ✅ Checkout
    ├── ✅ Setup Python
    ├── ✅ Install dependencies
    ├── ✅ Generate previews (45s)
    ├── ✅ Validate filenames (12s)
    ├── ✅ Generate index.html (8s)
    └── ✅ Commit results (5s)

Total: 1min 15s
```

Commits automatiques

Exemple de commit généré:

```
🤖 auto: previews, validation, index

- Generated 24 WebP previews
- Validated 156 images
- Updated index.html
```

✅ Points de vérification

- [ ] Fichier `.github/workflows/build-all.yml` créé
- [ ] Permissions `write` activées

- [ ] Premier workflow vert (✅) dans Actions
- [ ] Commits automatiques visibles

⚠ Erreurs fréquentes

Permission denied lors du push:

```
# Solution: Ajouter dans le job
permissions:
  contents: write
```

Workflow ne se déclenche pas:

- Vérifier les chemins dans `on.push.paths`
- Vérifier que les scripts sont à la racine
- Attendre 1-2 minutes (délai GitHub)

Erreur "No changes to commit":

- Normal si aucune modification
- Le `git diff --quiet || commit` gère ce cas

🤖 Étape 11 : Analyse IA avec GPT-4o Vision

Objectif

Utiliser GPT-4o Vision pour analyser automatiquement les images par lots, avec citations DTU et génération de résultats structurés (CSV, JSON, SVG)[1].

Architecture d'analyse en 3 passes

```
┌──────────────┐
│  Pass 0      │  Pré-analyse contextuelle (CSV rapide)
│  Contexte    │  → Identification zones, éléments visibles
└──────┬───────┘
       ↓

┌──────────────┐
```

```
│  Pass 1     │  Analyse détaillée (JSON + citations DTU)
│  Anomalies  │  → Défauts, causes, gravité, recommandations
└─────┬──────┘
      ↓
┌─────┴──────┐
│  Pass 2     │  Annotations visuelles (SVG overlays)
│  SVG        │  → Marqueurs sur zones défectueuses
└─────┬──────┘
```

Pass 0 : Pré-analyse CSV

Objectif: Contextualiser rapidement chaque image[1].

Prompt système:

```
Tu es un expert en diagnostic bâtiment.
Analyse l'image et génère UNIQUEMENT un CSV (pas de texte avant/après).

Format stricte:
filename,zone,elements_visibles,phase_travaux,conditions_meteo

Exemple:
0123_DETAIL_facade_20241015.jpg,facade,enduit+fissures+balcon,post-construction,sec
```

Commande d'utilisation:

```
Modèle: gpt-4o
Température: 0.1 (précision maximale)
Max tokens: 500

Entrées:
- url_preview (images WebP)
- Lot de 10-20 images max

Sortie:
CSV brut à sauvegarder dans json/context_pass0.csv
```

Exemple de sortie:

```
filename,zone,elements_visibles,phase_travaux,conditions_meteo
0123_facade_20241015.jpg,facade,enduit+fissures+balcon,post-construction,sec
2345_humidite.jpg,mur_interieur,traces_eau+moisissures+peinture,sinistre,humide
```

Pass 1 : Analyse détaillée JSON

Objectif: Détecter anomalies avec citations DTU précises[1].

Prompt système:

```
Tu es un expert diagnostic bâtiment avec accès aux DTU.

Documents disponibles:
<INSÉRER json/docs_index.json>

Analyse CHAQUE image et génère UNIQUEMENT un JSON (pas de texte).

Format stricte:
{
  "filename": "...",
  "anomalies": [
    {
      "type": "fissure_structurelle",
      "gravite": "critique",
      "description": "Fissure traversante 3mm, orientation 45°",
      "localisation": {"x": 0.45, "y": 0.62},
      "causes_probables": ["tassement différentiel", "retrait thermique"],
      "citations_dtu": [
        {
          "doc_id": "DTU_20.1",
          "pages": [15, 16],
          "extrait": "Fissures > 2mm nécessitent..."
        }
      ],
      "recommandations": ["Expertise structure", "Suivi mensuel"],
      "confidence": 0.85
```

```
    }
  ],
  "etat_general": "préoccupant",
  "incertitudes": ["Profondeur fissure non mesurable"]
}
```

Paramètres recommandés:

```
Modèle: gpt-4o
Température: 0.2
Max tokens: 4000

Entrées:
- url_original ou url_preview
- json/docs_index.json (pour citations)
- Lot de 5-10 images max

Sortie:
JSON à sauvegarder dans json/analysis_pass1.json
```

Exemple de sortie:

```
{
  "analyses": [
    {
      "filename": "2345_DETAIL_humidite.jpg",
      "anomalies": [
        {
          "type": "infiltration_eau",
          "gravite": "majeure",
          "description": "Auréole humide 40×60cm avec moisissures noires",
          "localisation": {"x": 0.35, "y": 0.55, "w": 0.25, "h": 0.30},
          "causes_probables": [
            "défaut étanchéité toiture",
            "condensation chronique"
          ],
          "citations_dtu": [
            {
              "doc_id": "DTU_43.1",
```

```
            "pages": [^23],
            "extrait": "L'étanchéité doit garantir..."
          }
        ],
        "recommandations": [
          "Recherche fuite urgente",
          "Traitement anti-moisissures",
          "Ventilation mécanique"
        ],
        "confidence": 0.92
      }
    ],
    "etat_general": "dégradé",
    "incertitudes": ["Source exacte infiltration"]
  }
 ]
}
```

Pass 2 : Overlays SVG

Objectif: Générer des annotations visuelles superposables sur les images[1].

Prompt système:

```
Génère un SVG avec annotations des zones défectueuses.

Format stricte:
<svg viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg">
  <!-- Rectangle zone anomalie -->
  <rect x="30" y="45" width="25" height="20"
        fill="none" stroke="red" stroke-width="0.5" opacity="0.8"/>

  <!-- Label -->
  <text x="32" y="43" font-size="3" fill="red">Fissure</text>

  <!-- Flèche -->
  <line x1="42.5" y1="55" x2="42.5" y2="70"
        stroke="red" stroke-width="0.3" marker-end="url(#arrow)"/>
```

```
</svg>
```

Utilisation des coordonnées:

```
Système de coordonnées normalisées:
- viewBox="0 0 100 100" (0-100% de l'image)
- x, y en pourcentage
- Facilite superposition sur images de tailles différentes
```

Exemple de sortie:

```
<svg viewBox="0 0 100 100" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <marker id="arrow" markerWidth="10" markerHeight="10" refX="5" refY="5">
      <polygon points="0,0 10,5 0,10" fill="red"/>
    </marker>
  </defs>

  <!-- Zone infiltration -->
  <rect x="35" y="55" width="25" height="30"
        fill="rgba(255,0,0,0.1)" stroke="red" stroke-width="0.5"/>
  <text x="37" y="53" font-size="3" fill="red" font-weight="bold">Infiltration</text>

  <!-- Zone moisissures -->
  <circle cx="45" cy="68" r="8"
          fill="rgba(0,0,255,0.1)" stroke="blue" stroke-width="0.5"/>
  <text x="41" y="80" font-size="2.5" fill="blue">Moisissures</text>
</svg>
```

Lotisation et contrôle

Commandes de contrôle dans le chat:

```
SUIVANT → Analyser les 10 images suivantes
STOP → Arrêter l'analyse, sauvegarder résultats
REJOUE → Relancer analyse d'une image spécifique
```

Exemple d'utilisation:

```
Prompt 1:
"Analyse Pass 1 pour images 1-10 de json/images_db.json"

Réponse IA:
[JSON avec 10 analyses]

Prompt 2:
"SUIVANT"

Réponse IA:
[JSON avec images 11-20]
```

Few-shot learning (exemples d'apprentissage)

Mauvais exemple (à éviter):

```
{
  "anomalies": [
    {
      "type": "dégradation",
      "description": "Ça a l'air abîmé",
      "gravite": "moyen",
      "recommandations": ["Réparer"]
    }
  ]
}
```

Bon exemple (à reproduire):

```
{
  "anomalies": [
    {
      "type": "fissure_structurelle",
      "gravite": "critique",
      "description": "Fissure traversante verticale, largeur 3-4mm, longueur 1.2m, du plancher au plafond",
      "localisation": {"x": 0.72, "y": 0.35, "w": 0.05, "h": 0.55},
```

```
      "causes_probables": [
        "tassement différentiel fondation",
        "retrait thermique béton",
        "surcharge structurelle"
      ],
      "citations_dtu": [
        {
          "doc_id": "DTU_20.1",
          "pages": [15, 16],
          "extrait": "Les fissures > 2mm traversantes nécessitent expertise structure
immédiate"
        }
      ],
      "recommandations": [
        "Expertise structure urgente (7 jours)",
        "Mise en place témoins fissurométriques",
        "Vérification fondations",
        "Interdiction d'usage si progression"
      ],
      "confidence": 0.87,
      "mesures_precision": {
        "largeur_mm": [3, 4],
        "longueur_cm": 120,
        "orientation_degres": 85
      }
    }
  ],
  "etat_general": "critique",
  "incertitudes": [
    "Profondeur réelle de la fissure (nécessite sondage)",
    "Ancienneté exacte (pas d'historique photo)"
  ]
}
```

✅ Points de vérification

- [ ] Température ≤ 0.2 (précision)
- [ ] docs_index.json chargé dans le prompt

- [ ] Sorties strictes (pas de narration)
- [ ] Champ `confidence` renseigné
- [ ] `UNKNOWN` utilisé si doute

⚠️ Limitations importantes

Pages DTU dans PDF scannés:

- PDF natif: Citations pages précises ✅
- PDF scanné: Pages en `UNKNOWN` ❌

Hallucinations IA:

- Toujours mettre `confidence` < 1.0
- Exiger liste `incertitudes`
- Vérifier citations DTU manuellement

🎯 Étape 12 : Contrôles qualité finaux

Objectif

Garantir la cohérence, lisibilité et traçabilité de l'ensemble du projet avant utilisation opérationnelle[1].

Checklist de vérification

1. Intégrité des fichiers

Images:

- [ ] Toutes les images respectent patterns A/B/C
- [ ] Aucun Id_File en doublon
- [ ] Aucune erreur dans CSV de validation
- [ ] Previews générés pour 100% des images

JSON:

- [ ] `json/images_db.json` valide (jsonlint.com)
- [ ] `json/docs_index.json` valide
- [ ] Tous les champs obligatoires présents
- [ ] URLs RAW fonctionnelles (test échantillon)

HTML:

- [ ] `index.html` s'affiche correctement
- [ ] Galerie complète (toutes images visibles)
- [ ] Liens vers previews fonctionnels

---

2. Cohérence des métadonnées

ROI (regions of interest):

```
# Vérifier échantillon dans images_db.json
{
  "roi_hints": {
    "x": 0.5,  // ✅ Entre 0.0 et 1.0
    "y": 0.5,  // ✅ Entre 0.0 et 1.0
    "w": 0.3,  // ✅ Entre 0.0 et 1.0
    "h": 0.3   // ✅ Entre 0.0 et 1.0
  }
}
```

Vérifications:

- [ ] $0 \leq x, y, w, h \leq 1$
- [ ] $x + w/2 \leq 1$ (ROI dans image)
- [ ] $y + h/2 \leq 1$ (ROI dans image)

---

3. Qualité des analyses IA

Échantillonnage:

- Sélectionner 5-10 images représentatives
- Vérifier analyses Pass 1 (JSON)

Critères de qualité:

Critère	Bon ✅	Mauvais ❌
Description	Précise, quantifiée (mm, cm)	Vague ("abîmé", "dégradé")
Localisation	Coordonnées x,y,w,h	Absente ou imprécise
Gravité	Échelle définie (mineur/majeur/critique)	Arbitraire
Causes	2-3 hypothèses plausibles	Liste exhaustive ou aucune
Citations	doc_id + pages précises	Absentes ou génériques
Confidence	0.0-1.0 avec justification	Toujours 1.0
Incertitudes	Listées explicitement	Absentes

---

4. Corrections manuelles

Ajuster ROI imprécis:

```
// Avant (ROI trop large)
{
  "filename": "2345_DETAIL_humidite.jpg",
  "roi_hints": {"x": 0.5, "y": 0.5, "w": 0.5, "h": 0.5}
}

// Après (ROI précis sur défaut)
{
  "filename": "2345_DETAIL_humidite.jpg",
  "roi_hints": {"x": 0.35, "y": 0.62, "w": 0.20, "h": 0.25}
}
```

Relancer génération crops:

```
python make_crops_from_roi.py
```

Relancer analyse IA:

```
Prompt: "Rejoue analyse Pass 1 pour image 2345_DETAIL_humidite.jpg avec nouveau crop"
```

---

5. Traçabilité et archivage

Manifest final:

```
# Générer checksums après tous les ajustements
python build_manifest.py

# Vérifier global_sha256
cat manifest.json | grep global_sha256
```

Historique Git:

```
# Vérifier commits
git log --oneline --graph

# Exemples de bons messages:
# ✅ feat: ajout 24 images facade chantier A
# ✅ fix: correction ROI images 2345-2350
# ✅ docs: mise à jour DTU 20.1 et 43.1
```

Sauvegarde locale:

```
# Archiver version finale
git archive --format=zip HEAD -o ArBot_v1.0_backup.zip
```

Tableau récapitulatif des fichiers clés

Fichier	Statut	Vérification

`images/**/*.jpg`	156 fichiers	✅ Noms valides
`images/preview/**/*.webp`	156 fichiers	✅ Générés
`images/crop/**/*_crop.jpg`	48 fichiers	✅ ROI corrects
`json/images_db.json`	1 fichier	✅ JSON valide
`json/docs_index.json`	1 fichier	✅ URLs RAW OK
`json/images_sidecar.csv`	1 fichier	✅ Aucun doublon
`json/analysis_pass1.json`	1 fichier	✅ Confidence OK
`index.html`	1 fichier	✅ Galerie fonctionnelle
`manifest.json`	1 fichier	✅ Hash global stable
`checksums_sha256.txt`	1 fichier	✅ Intégrité vérifiée

✅ Validation finale

Tous les points OK ?

- [ ] 0 lien cassé
- [ ] 0 image manquante
- [ ] 0 erreur de regex
- [ ] 0 doublon Id_File
- [ ] Analyses IA avec confidence et incertitudes
- [ ] Manifest à jour
- [ ] GitHub Pages accessible

→ Projet prêt pour utilisation opérationnelle 🎉

⚠ Erreurs critiques à éviter

Confondre hypothèse et constat:

```
X "La fissure est causée par un tassement" (affirmation)
✅ "Causes probables: tassement, retrait" + confidence 0.7
```

Oublier UNKNOWN:

```
X Inventer une page DTU
✅ "page": "UNKNOWN" + note dans incertitudes
```

Ignorer les incertitudes:

```
X confidence: 1.0 sans incertitudes listées
✅ confidence: 0.82 + incertitudes: ["profondeur non mesurable"]
```

📁 Étape 13 : Récapitulatif des fichiers clés

Objectif

Vue d'ensemble de tous les scripts, fichiers de configuration et données générés par le runbook[1].

Arborescence complète du projet

```
Arbot-MiniDB/
│
├── 📁 images/
│   ├── chantier/           # Images de chantier (Cas A)
│   ├── plans/              # Plans techniques (Cas B)
│   ├── sinistre/           # Images de sinistres (Cas C)
│   ├── preview/            # Previews WebP générés
│   └── crop/               # Crops ROI générés
│
├── 📁 docs/
│   ├── DTU_20.1_Maconnerie.pdf
│   ├── DTU_40.11_Couverture.pdf
│   └── Notice_*.pdf
│
├── 📁 json/
```

```
│   ├── images_db.json          # Index principal des images
│   ├── images_sidecar.csv      # Export CSV validation
│   ├── docs_index.json         # Catalogue des documents
│   ├── context_pass0.csv       # Résultats Pass 0
│   └── analysis_pass1.json     # Résultats Pass 1
│
├── 📁 tiles/                   # (Optionnel) Tuiles 1024px
│   ├── *_tile_00.jpg
│   └── tiles_map.json
│
├── 📁 scripts/
│   ├── generate_checksum.bat   # Checksums Windows
│   └── generate_docs_index.py  # Index documents
│
├── 📁 .github/
│   └── workflows/
│       └── build-all.yml       # Automatisation CI/CD
│
├── 📄 validate_pack.py          # Validation noms + CSV/JSON
├── 📄 make_previews_and_augment_json.py  # Previews WebP
├── 📄 generate_index_html.py    # Galerie HTML
├── 📄 make_crops_from_roi.py    # Crops depuis ROI
├── 📄 tile_1024_overlap.py      # Tuilage images
├── 📄 build_manifest.py         # Manifest + hash global
│
├── 📄 index.html                # Galerie publique
├── 📄 manifest.json             # Manifest intégrité
├── 📄 checksums_sha256.txt      # Checksums SHA-256
│
└── 📄 README.md                 # Documentation projet
```

Scripts Python (racine)

Script	Fonction	Dépendances	Sortie
`validate_pack.py`	Valide noms, unicité, cohérence Id_File	Aucune	CSV + JSON

`make_previews_and_augment_json.py`	Génère previews WebP + enrichit JSON	Pillow	WebP + JSON
`generate_index_html.py`	Crée galerie HTML	Aucune	index.html
`make_crops_from_roi.py`	Extrait crops depuis ROI	Pillow	JPG crops
`tile_1024_overlap.py`	Découpe images en tuiles	Pillow	Tuiles + JSON
`build_manifest.py`	Génère manifest + hash global	hashlib	manifest.json

Scripts utilitaires (scripts/)

Script	Fonction	Usage
`generate_checksum.bat`	Checksums SHA-256 (Windows)	`scripts\generate_checksum.bat`
`generate_docs_index.py`	Index des documents PDF	`python scripts\generate_docs_index.py`

Fichiers de données (json/)

Fichier	Format	Contenu	Généré par
`images_db.json`	JSON	Index principal images avec métadonnées	`validate_pack.py` + `make_previews...`
`images_sidecar.csv`	CSV	Export validation (Excel-friendly)	`validate_pack.py`
`docs_index.json`	JSON	Catalogue documents PDF avec URLs	`generate_docs_index.py`

`context_pass0.csv`	CSV	Contexte pré-analyse	GPT-4o Pass 0
`analysis_pass1.json`	JSON	Analyses détaillées + citations	GPT-4o Pass 1

Fichiers web

Fichier	Type	Description	URL
`index.html`	HTML	Galerie interactive	`https://tazevil.github.io/Arbot-MiniDB/`

Fichiers de traçabilité

Fichier	Format	Contenu
`manifest.json`	JSON	Hash SHA-256 de chaque fichier + hash global
`checksums_sha256.txt`	TXT	Checksums en format texte brut

Configuration CI/CD

Fichier	Type	Fonction
`.github/workflows/build-all.yml`	YAML	Workflow automatisation GitHub Actions

Commandes récapitulatives

Installation dépendances:

```
pip install pillow
```

Workflow complet (exécution manuelle):

```
# 1. Validation
python validate_pack.py --images .\images --out .\json --owner Tazevil --repo
Arbot-MiniDB --branch main

# 2. Previews
python make_previews_and_augment_json.py --images .\images --out_json
.\json\images_db.json --owner Tazevil --repo Arbot-MiniDB --branch main

# 3. Galerie
python generate_index_html.py --images .\images --out .\index.html

# 4. Index documents
python .\scripts\generate_docs_index.py

# 5. Crops (optionnel)
python make_crops_from_roi.py

# 6. Tuilage (optionnel)
python tile_1024_overlap.py --src .\images --out .\tiles --size 1024 --overlap 0.10

# 7. Manifest
python build_manifest.py

# 8. Publication
git add .
git commit -m "feat: workflow complet"
git push
```

🎓 Annexe : Concepts clés pour débutants

Concepts techniques

1. Expression régulière (Regex):

Patron de recherche pour valider formats de texte

```
Pattern: (\d{4})_DETAIL_(\w+)\.jpg
Valide: 0123_DETAIL_facade.jpg ✅
Invalide: 123_DETAIL_facade.jpg X (3 chiffres)
```

2. Hash SHA-256:

Empreinte numérique unique d'un fichier (64 caractères hexadécimaux)

```
Fichier → SHA-256 → a3f5c8d9e2b1f7a6...
Modification → SHA-256 → b9c2d4e6f8a1b3c5... (différent)
```

3. GitHub Pages:

Hébergement gratuit de sites web statiques (HTML/CSS/JS) sur GitHub

4. URLs RAW:

URLs directes vers fichiers bruts sur GitHub (sans interface web)

```
https://raw.githubusercontent.com/USER/REPO/BRANCH/path/file.jpg
```

5. CI/CD (Continuous Integration/Deployment):

Automatisation tests et déploiements à chaque modification code

---

Formats de fichiers

Format	Extension	Usage ArBot	Avantages
JPEG	`.jpg`	Images originales	Universel, léger
WebP	`.webp`	Previews	30% plus léger que JPEG
PNG	`.png`	Schémas, plans	Transparence, sans perte
CSV	`.csv`	Exports validation	Compatible Excel
JSON	`.json`	Métadonnées, config	Structuré, facile à parser
SVG	`.svg`	Annotations vi	

⁂

---

1. ArBot-Runbook-complet-sequence-et-operationnel.pdf