

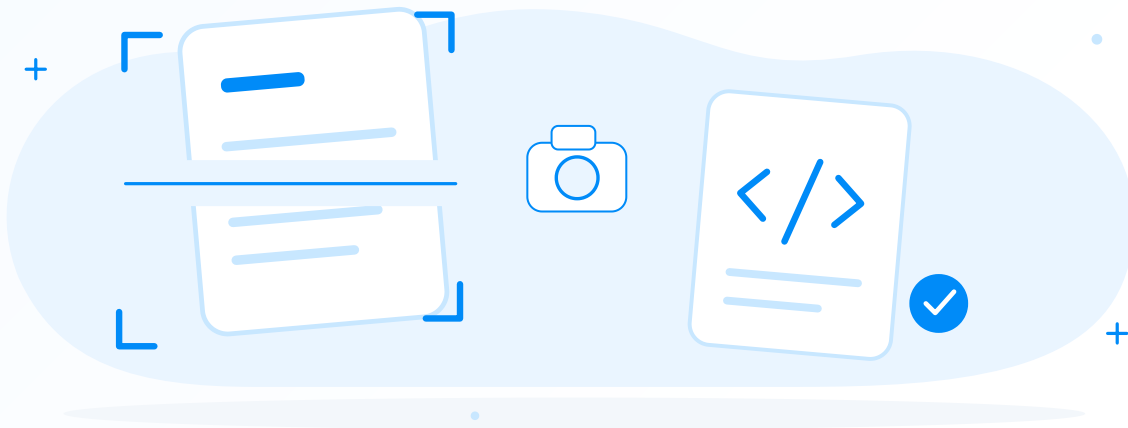
Tutoriel

Multimodal · OCR · Extraction

Extraire des informations d'un document avec un modèle vision

Analyser une image de facture avec un modèle multimodal et valider les champs extraits avant de les enregistrer.

Auteur **Youness Boutyour**

**DURÉE**

55 minutes

NIVEAU

Intermédiaire

FORMAT

Tutoriel pratique

Multimodal · OCR · Extraction · publié le 2026-07-22 · mock

Objectifs et livrables

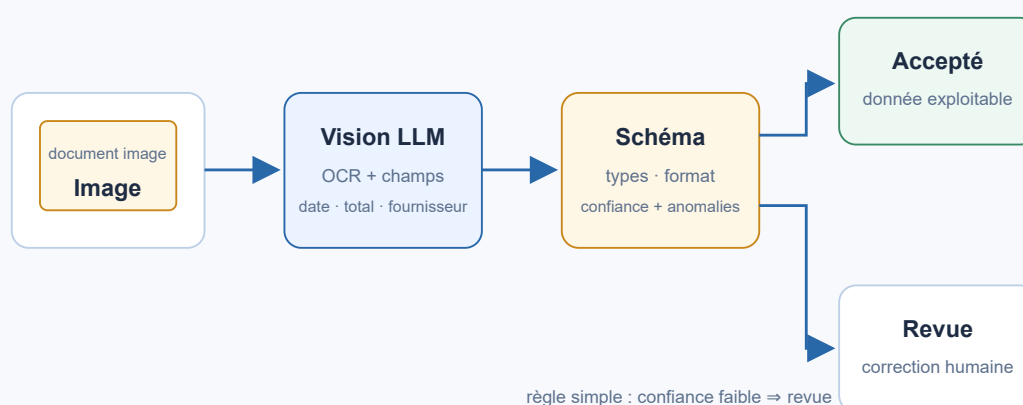
Une facture reçue sous forme d'image doit souvent être ressaisie avant de pouvoir être traitée. Nous allons automatiser une partie de ce travail : demander à un modèle vision de proposer les champs utiles, puis vérifier ces valeurs en Python.

Le tutoriel s'appuie sur deux exemples de facture pour suivre le parcours d'une image jusqu'à une sortie structurée. Vous préparerez la consigne d'extraction, récupérerez la réponse du modèle et contrôlerez les champs, les types, la date et le montant. Nous examinerons ensuite les invariants métier et le traitement d'un document difficile à lire.

Vous verrez pourquoi il est utile de conserver l'image et la réponse brute à côté des valeurs validées. Lorsqu'une extraction échoue, ces éléments permettent de retrouver le champ en cause et de décider si une nouvelle image ou une vérification manuelle est nécessaire.

Le programme produira `outputs/invoice.json` pour une extraction acceptée ou `outputs/review.json` pour un cas à examiner. Vous disposerez ainsi d'un exemple complet de préremplissage contrôlé, avec un résultat que l'on peut relire avant de l'intégrer à un autre système.

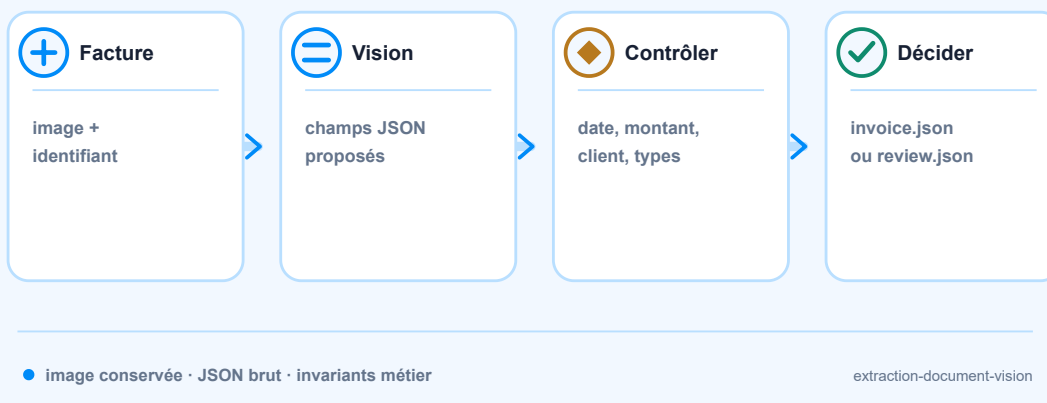
Figure — l'image n'est qu'une entrée : la validation décide si la donnée est exploitable



Chaîne d'extraction et de validation des champs de facture.

Transformer une image en donnée contrôlée

Schéma de l'extraction de facture : image, proposition JSON, validation des types et décision d'acceptation ou de revue.



La sortie du modèle devient une donnée utilisable seulement après validation des types et invariants.

Lire le flux avant de coder

Le schéma place la validation après l'extraction. Le modèle vision propose des champs à partir de l'image, mais il ne connaît pas nécessairement les contraintes de votre système : un montant doit être numérique, une date doit être interprétable et certains champs doivent être présents ensemble. La réponse brute reste donc une preuve de travail, pas le livrable final.

Une extraction acceptable produit invoice.json ; un doute ou une incohérence produit review.json avec assez de contexte pour une reprise humaine. Cette séparation évite de masquer une erreur de lecture en la remplaçant par une valeur par défaut.

1. Préparer l'environnement

Le contrôle Pillow vérifie qu'un fichier image peut être décodé ; il ne confirme ni la lisibilité du texte ni l'exactitude de son contenu. Lancez-le après avoir placé les fixtures dans `data/`. En cas d'échec, vérifiez d'abord le fichier et son format avant de modifier le prompt ou le schéma : le modèle n'a pas encore été sollicité à cette étape.

Avant tout appel multimodal, vérifiez que Python peut ouvrir l'image de test. `Path.is_file` contrôle le chemin et Pillow examine le format du fichier ; une image qui passe ces tests peut néanmoins contenir du texte difficile à lire. Ce contrôle technique prépare donc l'extraction sans encore juger la qualité du document.

L'appel `image.verify()` recherche des problèmes de fichier sans réaliser la lecture des champs de facture. Exécutez-le une fois les images de l'étape suivante placées dans `data/`. Une erreur à ce stade indique un chemin ou un fichier à corriger, ce qui évite de diagnostiquer à tort un problème du modèle.

Terminal · Bash

```
mkdir -p extraction-document-vision/{data,src,outputs}
cd extraction-document-vision
python3 -m venv .venv && source .venv/bin/activate
python -m pip install jsonschema pillow
```

Sortie obtenue

Sortie / exemple

```
Successfully installed jsonschema pillow
```

Commande de vérification

Terminal · Bash

```
python -c "from PIL import Image; print('vision stack ok')"
```

Code de l'étape

Contrôlez l'existence et le format avant d'appeler le modèle :

Python

```
from pathlib import Path
from PIL import Image

image_path = Path("data/invoice-clear.jpg")
assert image_path.is_file()
with Image.open(image_path) as image:
    image.verify()
print(image_path, image_path.stat().st_size, "octets")
```

2. Préparer deux fixtures

Regardez les deux images avant l'extraction et notez manuellement les champs que vous pouvez lire. Cette petite référence permettra de repérer une date plausible mais incorrecte ou un montant confondu avec un autre. L'empreinte affichée sert à reconnaître la fixture utilisée ; un préfixe court de SHA-256 est

un repère de lecture, pas un identifiant dont l'unicité serait garantie.

Les deux fixtures servent à comparer des comportements, pas seulement à vérifier que deux fichiers existent. Ouvrez-les visuellement pour repérer le fournisseur, la date et le montant qui devraient être retrouvés. Notez les zones ambiguës : elles expliqueront ensuite pourquoi une extraction doit être relue.

Le hash SHA-256 identifie le contenu du fichier utilisé pour l'essai. Deux images portant le même nom peuvent différer après un recadrage ou une compression ; le hash permet de retrouver cette différence. Les noms et dimensions des sorties illustratives peuvent varier selon les fixtures utilisées : comparez d'abord les fichiers réellement inspectés et leur rôle dans le test.

Placez `data/invoice-clear.jpg` et `data/invoice-review.jpg`. La première est un document de test OCR sous licence CC0 ; la seconde est une fixture de type document nécessitant une revue humaine.

Sortie obtenue

Les deux images sont présentes et leur taille est enregistrée.

Sortie / exemple

```
invoice-clear.jpg width=1600 height=1100
invoice-blurry.jpg width=640 height=440
```

Commande de vérification

Terminal · Bash

```
python src/inspect.py data/*.jpg
```

Code de l'étape

Python

```
import hashlib
from pathlib import Path

for image_path in Path("data").glob("*.jpg"):
    digest = hashlib.sha256(image_path.read_bytes()).hexdigest()
    print(image_path.name, image_path.stat().st_size, digest[:12])
```

3. Demander puis valider l'extraction

La conversion en base64 transporte les octets de l'image dans la requête ; elle n'améliore pas sa qualité et ne constitue pas une reconnaissance de caractères. Le type annoncé doit correspondre au fichier transmis. Après la réponse, examinez la structure puis les valeurs : un document correctement sérialisé en JSON peut encore contenir une date inventée ou un chiffre mal lu.

Le texte de la demande et l'image sont envoyés dans un même message. Le Base64 transporte les octets de l'image sous forme de texte ; le préfixe de la data URL indique ici un contenu JPEG. Cette conversion ne corrige ni le flou ni les valeurs mal lisibles.

Après l'appel, le programme doit d'abord décoder le JSON, puis vérifier les champs et leurs types. L'assertion de l'extrait vérifie uniquement la présence des clés : elle ne remplace pas le schéma complet. Enfin, la valeur `confidence` est une estimation produite par le modèle ; le seuil de démonstration doit être confronté à des documents annotés avant de servir de règle d'acceptation réelle.

Encodez l'image, demandez uniquement `supplier`, `date`, `total` et `confidence`, puis validez la sortie avec un schéma. Si `confidence < 0.85`, écrivez une revue plutôt qu'une facture acceptée.

Sortie obtenue

La facture nette produit un JSON valide ; la facture floue est orientée vers la revue.

Sortie / exemple

```
invoice-clear.json schema=valid confidence=0.97
invoice-review.json schema=valid confidence=0.62 action=human_review
```

Le mode réel encode l'image en data URL et appelle un modèle vision compatible avec `python src/extract.py --real data/invoice-clear.jpg`. Vérifiez toujours le schéma avant import.

Code de l'étape

Python

```
import base64, json
from pathlib import Path
from llm_client import chat

encoded = base64.b64encode(Path(image_path).read_bytes()).decode()
raw = chat([{"role": "user", "content": [
    {"type": "text", "text": "Extrait supplier, date, total et confidence en JSON uniquement."},
    {"type": "image_url", "image_url": {
        "url": f"data:image/jpeg;base64,{encoded}"
    }},
]]),
```

```
}})  
payload = json.loads(raw)  
assert {"supplier", "date", "total", "confidence"} ≤ set(payload)
```

Commande de vérification

Terminal · Bash

```
python src/extract.py --mock data/invoice-clear.jpg  
python src/extract.py --mock data/invoice-review.jpg
```

4. Vérifier les invariants métier

Les règles métier complètent la validation de forme : une date analysable et un total positif restent à comparer au document. Vérifiez les types avant les comparaisons numériques et lisez le montant dans son contexte, notamment les séparateurs décimaux. Le seuil de confiance trie ici les sorties pour la revue ; il ne remplace pas cette confrontation avec l'image et ne prouve pas la calibration du modèle.

Les contrôles métier portent sur les valeurs extraites. `date.fromisoformat` vérifie qu'une chaîne représente une date lisible au format attendu, mais ne prouve pas que cette date correspond à la facture. De même, un montant positif peut avoir été mal lu. Confrontez les sorties acceptées aux champs visibles sur l'image.

Le choix entre `invoice.json` et `review.json` rend la décision explicite pour le traitement suivant. Vérifiez la validité des types avant les comparaisons numériques, puis examinez la raison d'un refus. Pour un document envoyé en revue, conservez assez d'éléments pour permettre la correction sans devoir relancer aveuglément l'extraction.

Contrôlez la date ISO, le total positif et la présence du fournisseur. Conservez l'image, le hash et la réponse brute dans un dossier non publié.

Sortie obtenue

Le JSON accepté passe les invariants et la réponse de revue ne peut pas être importée par erreur.

Sortie / exemple

```
invoice-clear status=accepted total=128.40 currency=EUR  
invoice-blurry status=review reason="confidence below threshold"
```

Commande de vérification

Terminal · Bash

```
python src/quality.py outputs
python -m json.tool outputs/invoice.json >/dev/null
python -m json.tool outputs/review.json >/dev/null
```

Code de l'étape

Python

```
from datetime import date

confidence = float(payload["confidence"])
valid_date = date.fromisoformat(payload["date"])
accepted = bool(payload["supplier"]) and payload["total"] > 0 and confidence ≥ 0.85
output_name = "outputs/invoice.json" if accepted else "outputs/review.json"
print(output_name, valid_date.isoformat())
```

Exemples de sorties

Une image est acceptée et une image est envoyée en revue. Le mode réel peut varier selon le modèle et la qualité du scan ; comparez toujours avec la vérité terrain.

Dépannage et diagnostics

Réduisez la taille de l'image si l'API refuse le payload. Ne stockez pas de factures réelles dans le dépôt et masquez les données personnelles dans les exemples.

Vérification finale

- ☐ champs structurés ;
- ☐ confiance contrôlée ;
- ☐ image floue refusée ;
- ☐ réponse brute et hash conservés.

Limites et prolongements

Le modèle peut lire une valeur erronée avec assurance. Ajoutez une double validation humaine pour les montants et une mesure par type de document.

Références

- [OpenAI — Vision guide](#)
- [Python — base64](#)
- [Wikimedia Commons — Test OCR document.jpg \(CC0 1.0\)](#)