

# Reconnaître des algorithmes manuscrits et les traduire en code Python

Réalisé par :

Hamza BOUAZZA | Oussama BOUSLIM | Aya BELHADJI  
Wijdane HROUR | Salma ERRAJI CHAHID | Aymen RABBAH

```

def collatz_sequence(n):
    # Initialisation du compteur
    compteur = 0

    # Boucle jusqu'à ce que N soit égal à 1
    while n > 1:
        if n % 2 == 0:
            # N est pair
            n = n // 2
        else:
            # N est impair
            n = n * 3 + 1
        # Incrémentation du compteur
        compteur += 1

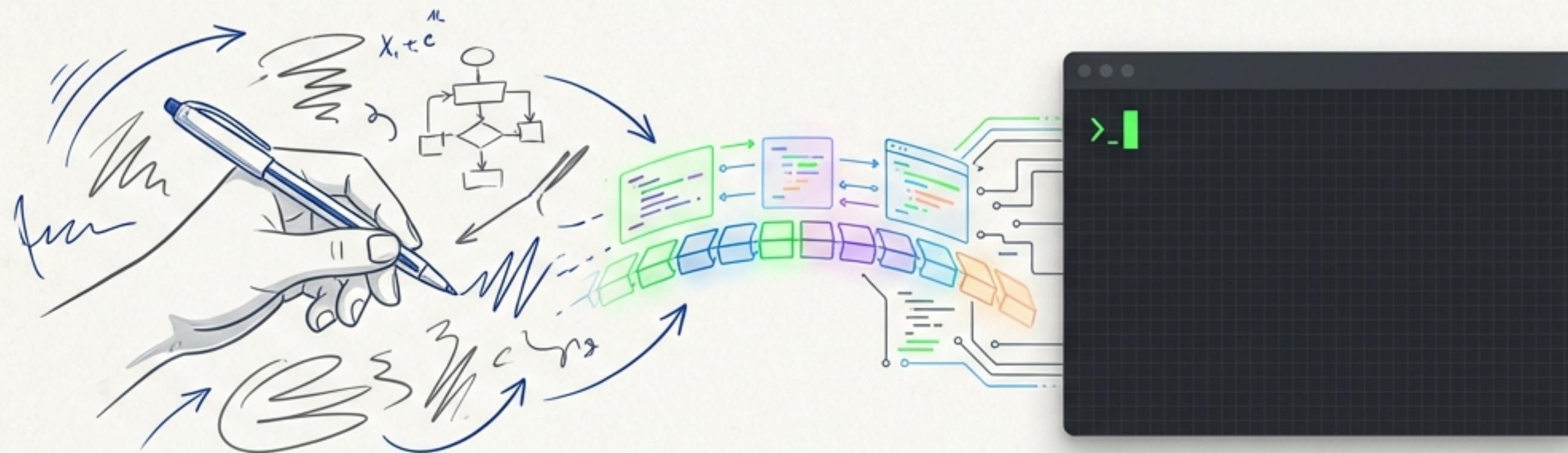
    # Affichage du résultat
    print(f"Nombre d'étapes : {compteur}")
    return compteur

# Exemple d'utilisation
if __name__ == "__main__":
    N_initial = 6
    resultat = collatz_sequence(N_initial)

```

Encadré par : M. Outman EL HICHAMI  
Module : Deep Learning

# Le Fossé entre l'Intuition et l'Exécution



## The Problem

L'écriture manuscrite est le moyen le plus intuitif pour concevoir des algorithmes, mais son transfert vers un ordinateur reste un processus manuel et lent.

## The Mission

Notre objectif est d'automatiser cette passerelle : développer une solution capable de lire un **pseudo-code manuscrit**, d'en comprendre la **syntaxe** et de générer un **code exécutable**.

```
def auto_bridge(handwritten_input):  
    return python_code
```

# La Matière Première : Un Dataset Complexé

## Description :

Utilisation d'un dataset public comprenant des milliers d'images d'algorithmes associées à leurs transcriptions (CSV).

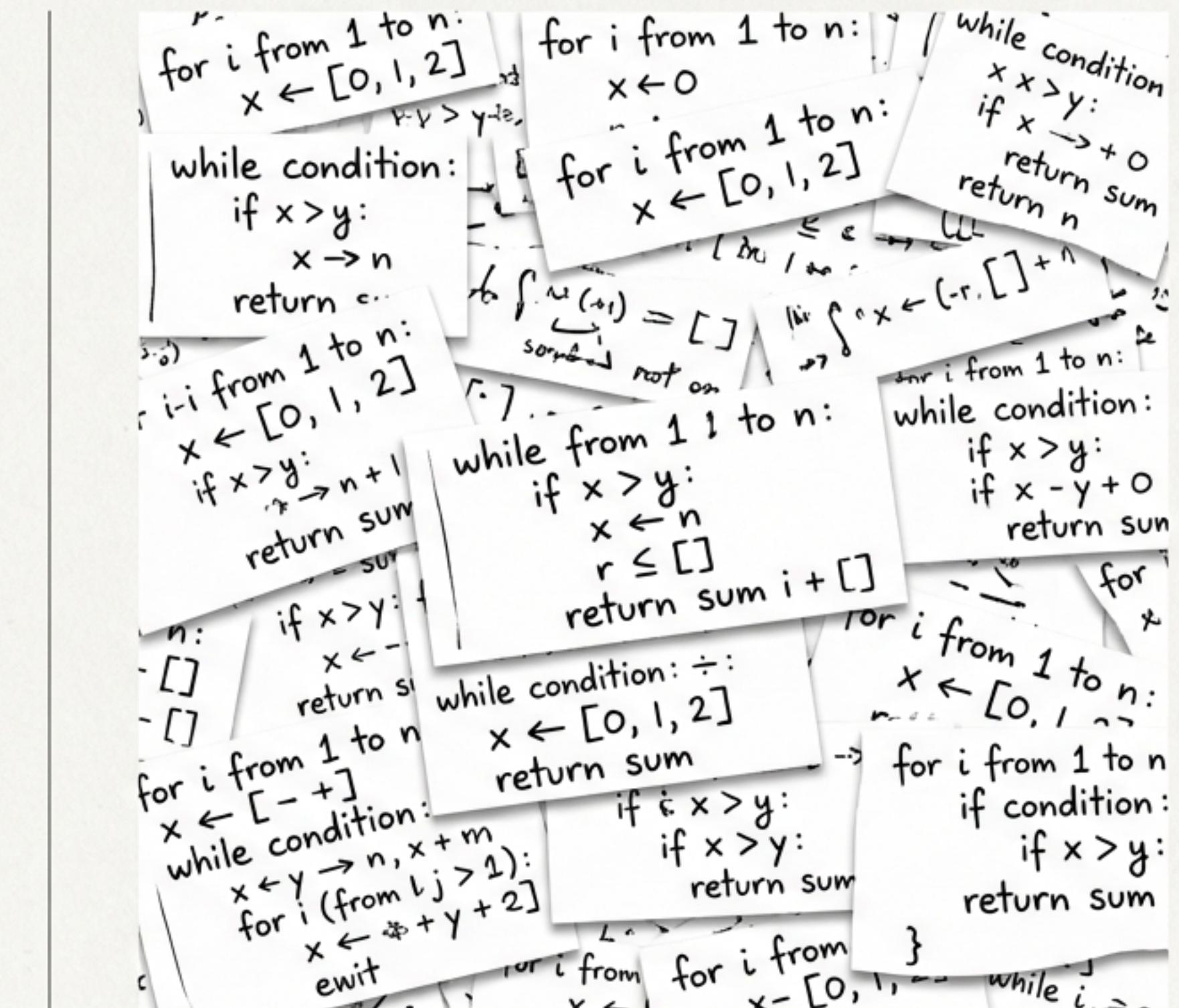
## Les Défis :

### 1. Variabilité Cursive

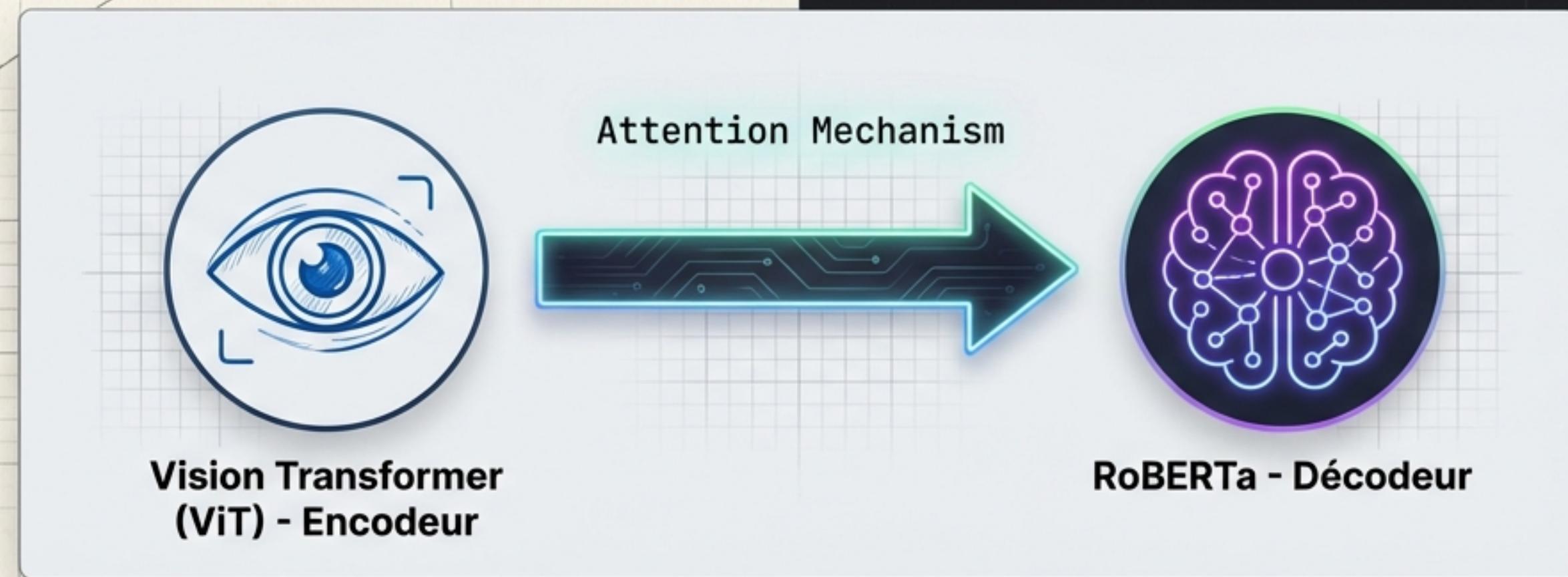
Chaque écriture est unique, rendant la standardisation difficile.

### 2. Symboles Spécifiques

Le modèle doit distinguer des éléments non-alphabétiques critiques comme les flèches d'affectation ( $\leftarrow$ ), les crochets ( $[]$ ) et les opérateurs mathématiques.



# L'Architecture : TrOCR (Transformer-based OCR)



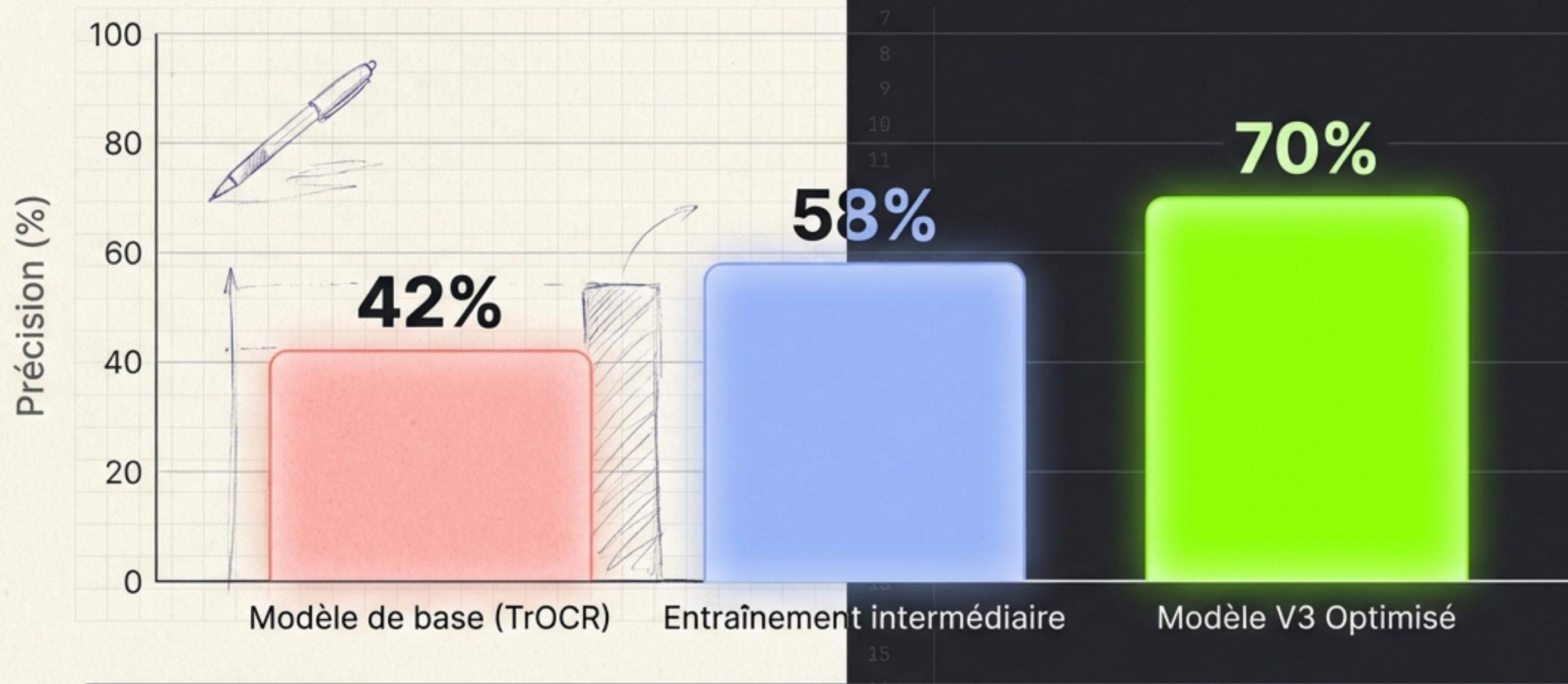
## Technologie Principale :

Nous avons sélectionné l'architecture TrOCR, qui combine la vision par ordinateur et le traitement du langage naturel.

## Pourquoi les Transformers ?

Contrairement aux méthodes CNN classiques, les **Transformers** (ViT + RoBERTa) offrent une robustesse supérieure pour comprendre le contexte séquentiel du code, ce qui est crucial pour respecter une syntaxe stricte.

# L'Évolution de la Précision (1 - CER)



## Résultat :

Une approche itérative a permis de transformer un modèle initial aux nombreuses lacunes en une solution V3 robuste, capable de gérer la syntaxe informatique.

rs = p min[1, .3]

# L'Usine Logicielle : Vue d'ensemble du danière e du Pipeline

Notre solution ne s'arrête pas à la reconnaissance ; c'est un pipeline complet en 4 étapes pour garantir un code exécutable.



## 1. Prétraitement Visuel

OpenCV / Otsu



## 2. Inférence Deep Learning

Modèle V3 (TrOCR)



## 3. Post-traitement Sémantique

Regex Rules



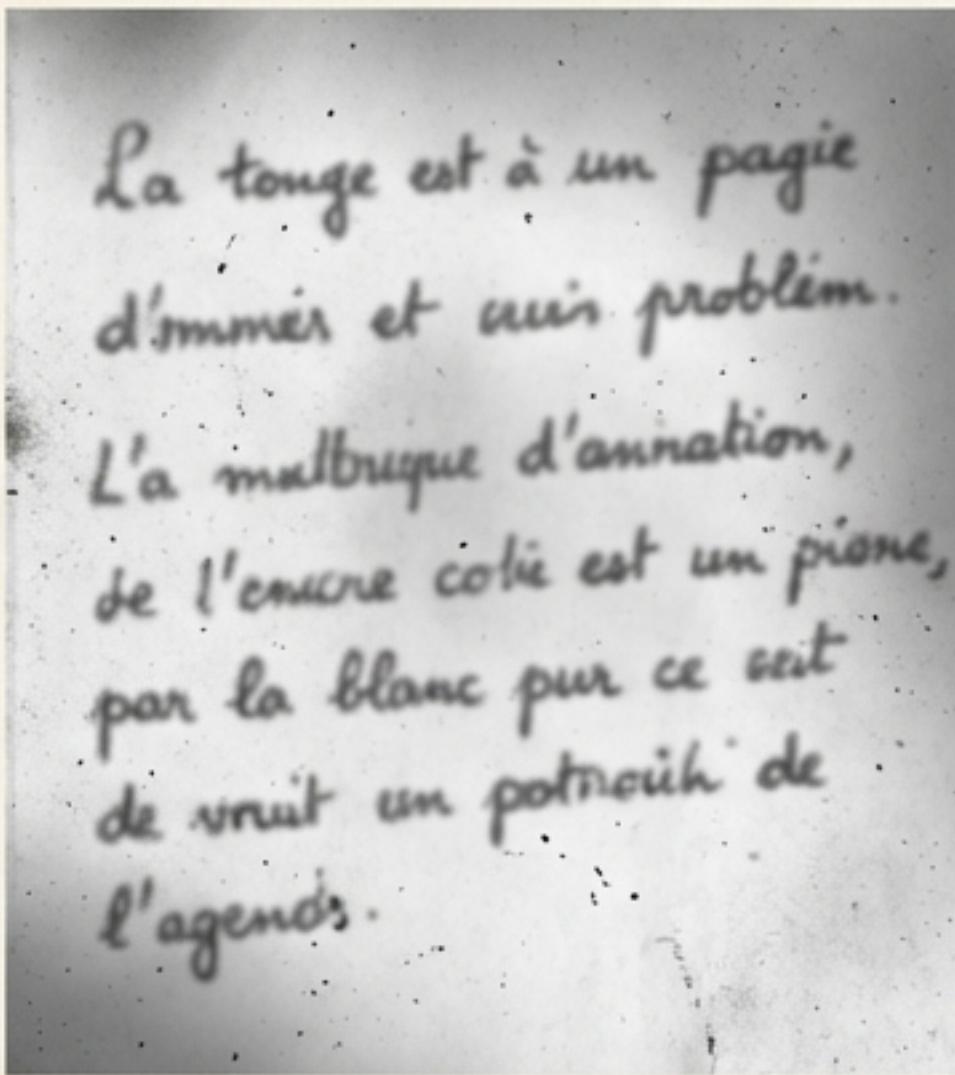
## 4. Transpilation Automatique

Python Script

# Étape 1 : Prétraitement Visuel

Nettoyer le canvas avant l'analyse

Avant



Après : Binarisation d'Otsu

La tongue est à un pagie d'ommier et auis problem. L'a malbrueque d'annation, de l'encre colie est un piane, par la blanc pur ce seit de vruit un potrouih de l'agendr.

2

Outils :

OpenCV + Binarisation d'Otsu

Processus :

- Transformation de l'image en noir et blanc pur.
- Élimination du bruit (tâches, ombres, imperfections du papier).

Objectif :

Permettre au modèle de se concentrer uniquement sur les traits de l'encre, sans distraction visuelle.

# Étape 2 : Inférence Deep Learning

Moteur : TrOCR V3 Optimisé



## 1. L'Œil (Vision Transformer - ViT) :

Analyse les pixels de l'image nettoyée par patches pour capturer la structure visuelle.

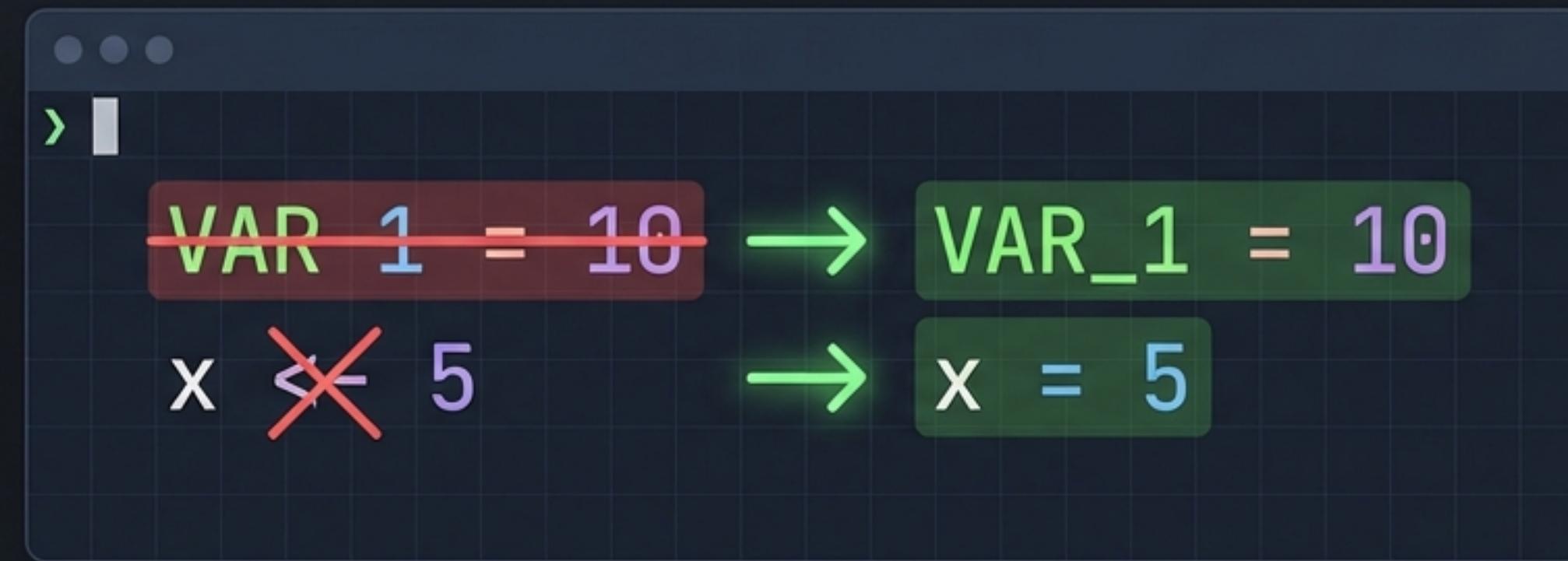
## 2. Le Cerveau (RoBERTa) :

Le décodeur linguistique prédit le texte brut correspondant à l'algorithme manuscrit.

Statut de sortie : Texte brut (Raw Text) contenant potentiellement des erreurs de syntaxe mineures.

# Étape 3 : Post-traitement Sémantique

La logique humaine au service de l'IA



**Outil** : Expressions Régulières (Regex)

**Problème** : L'IA peut commettre des erreurs sur des symboles proches visuellement.

Corrections Automatiques :

- **Correction de Variable** : `'VAR 1'` → `'VAR_1'` (Standardisation du format).
- **Reconstruction de Symboles** : Reconstruction des flèches d'affectation mal formées.

# Étape 4 : Transpilation vers Python

SI  $x > 0$  ALORS  
    AFFICHER 'Positif'  
SINON  
    AFFICHER 'Négatif'

si / sinon → if / else

```
1 • if x > 0:  
2     print('Positif')  
3 • else:  
4     print('Négatif')
```

**Mécanisme :** Script de traduction syntaxique

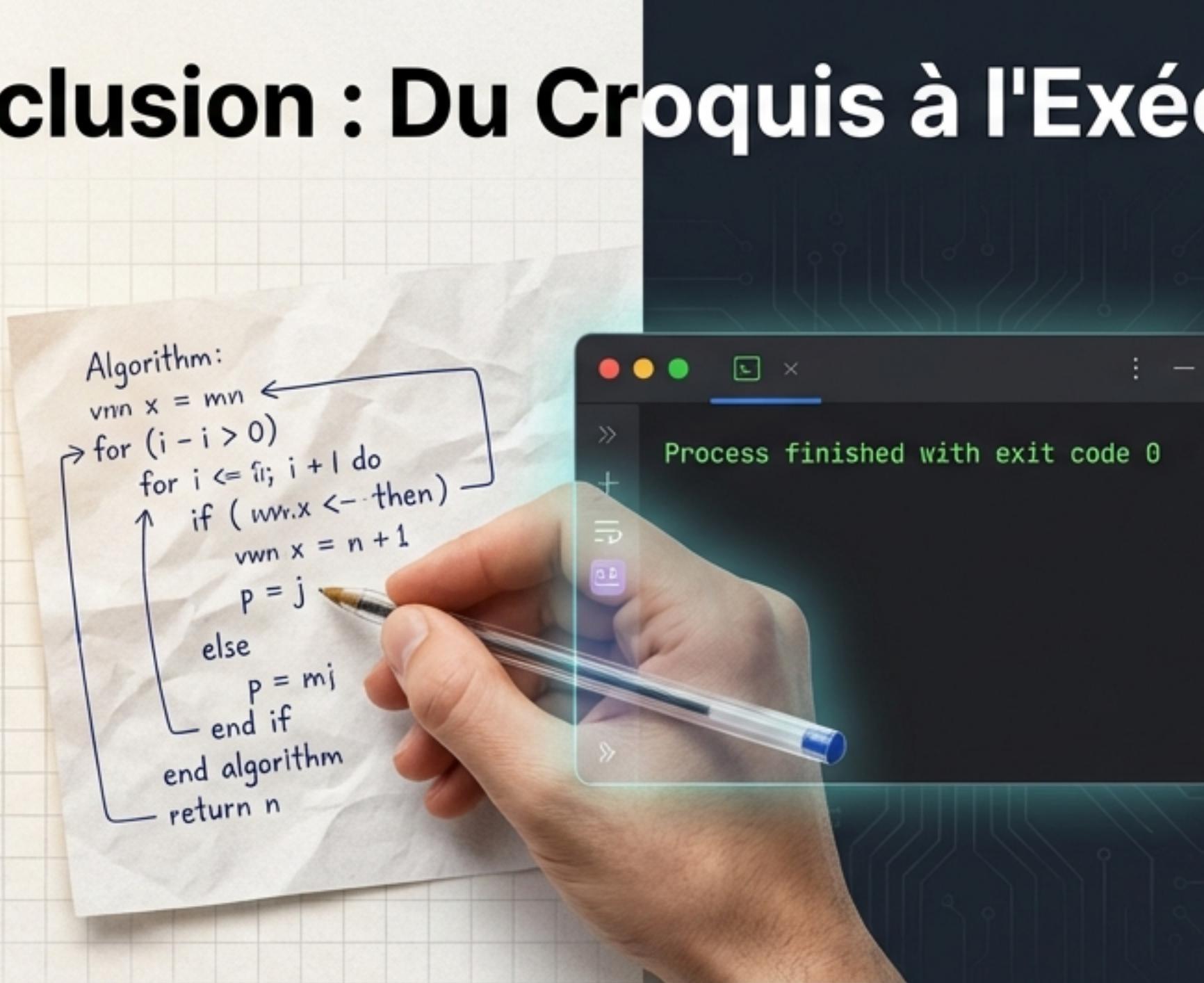
**Règles de Mapping :**

- 'si / sinon' → 'if / else'
- 'pour' → boucle 'for'
- 'tant que' → boucle 'while'
- Gestion automatique de l'indentation (cruciale pour Python).

**Indentation**

Gestion automatique de l'indentation  
(cruciale pour Python).

# Conclusion : Du Croquis à l'Exécution



Vision par Ordinateur + Transformers + Règles de Post-traitement = Code Fonctionnel

Ce pipeline prouve la faisabilité de digitaliser des algorithmes complexes de bout en bout.  
Impact : Passage d'un croquis sur papier à un programme Python fonctionnel en quelques secondes.