

ARCHITECTURE SCIENTIFIQUE DE LORACRAFT

De l'image à la traduction argumentée

Modèle de données • provenance • modules • corpus



Didier DTL Morandi

Version 1.1 • 17 août 2026

Statut du document

Ce document décrit l'architecture scientifique cible du projet Loracrafft et l'état expérimental constaté au 17 août 2026. Il distingue :

- le **socle expérimental existant**, constitué de LoraLayout, de LoraGlyph 1.0.1, du noyau LoraDocument 1.0, de leurs adaptateurs et d'un workflow Phase 1 exécutable ;
- les **contrats d'architecture** que tout composant doit respecter ;
- les **modules projetés**, dont l'implémentation reste à réaliser ou à consolider.

Les exemples JSON sont normatifs quant à la séparation des couches et illustratifs quant aux champs encore susceptibles d'évoluer. Toute évolution incompatible d'un contrat devra faire l'objet d'une nouvelle version de schéma et d'une procédure de migration.

La règle de versionnement du projet est la suivante : une nouvelle famille de logiciel, de schéma, de catalogue ou de manifeste commence à 1.0. Une expérimentation incomplète est signalée par son statut et par ses limites documentées, non par un numéro de version inférieur à 1.

Révision du 17 août 2026

Cette révision intègre les résultats obtenus depuis le 29 juillet 2026, date de lancement de la phase de codage du projet :

- implémentation et validation du schéma LoraDocument 1.0 ;
- séparation effective de layout_objects, glyph_hypotheses et human_validations ;
- identifiants stables, écriture atomique, provenance d'image et validations multi-annotateurs append-only ;
- orchestration locale LoraImage → LoraLayout → LoraGlyph → LoraDocument sous Windows 11 et Python 3.12 ;
- redressement automatique par transformée de Hough et estimation circulaire modulo 90°, validé sur une stèle inclinée d'environ 44,7° (cf. test_5.png) ;
- constat que le découpage en composantes connexes ne correspond pas au découpage en signes ou cadrats ;
- adoption d'une stratégie **registres d'abord**, puis cadrats, puis signes ;
- constitution d'une base locale (KB) issue des ressources de Raymond Monfort (<http://sethy1.free.fr/>), contenant 6 722 fichiers glyphiques à inventorier comme variantes et non comme 6 722 signes Gardiner distincts ;
- expérimentation de la comparaison morphologique, notamment par les sept invariants de Hu, sans les considérer comme un reconnaisseur suffisant à eux seuls ;
- gel de test_2 comme cas de non-régression : test_3 reste interdit tant que chaque unité hiéroglyphique de test_2 n'est pas correctement délimitée ;

- archivage obligatoire des sorties par version et par image test, sans écrasement.

Les résultats expérimentaux cités dans cette révision sont des observations de développement. Ils ne constituent pas encore des métriques scientifiques généralisables à un corpus représentatif.

Table des matières

1. Vision et objectifs du projet Loracraftt	6
2. Principes scientifiques	10
3. Architecture générale.....	14
4. Modèle de données : LoraDocument	17
5. Description détaillée de chaque module	21
6. Cycle de vie d'une inscription	39
7. Gestion des hypothèses et de l'incertitude	41
8. Traçabilité et provenance	43
9. Architecture des corpus	45
10. LoraSDK	47
11. Interfaces publiques : API internes	53
12. Formats JSON.....	68
13. Charte de nommage	71
14. Scénarios complets : de la photo à la traduction	79
15. Feuille de route de développement	81
16. Conclusion.....	87
Annexes	88
Contributeurs	102

Pourquoi l'égyptologie a besoin d'une architecture rigoureuse

Les inscriptions égyptiennes sont rarement des objets textuels simples et définitivement établis. Elles nous parviennent sur des supports altérés, fragmentaires, remployés ou photographiés dans des conditions inégales. Un signe peut être incomplet, confondu avec un détail du relief ou recevoir plusieurs identifications plausibles. Le sens de lecture dépend de la composition, de l'orientation des figures et parfois du contexte monumental. À ces difficultés matérielles s'ajoutent celles de la langue : graphies variables, valeurs multiples d'un même signe, simplifications, omissions, inversions honorifiques, changements historiques et traditions éditoriales concurrentes.

Une lecture publiée n'est donc pas une transcription mécanique de ce qui serait immédiatement visible. Elle résulte d'une suite de choix : délimiter une région, rapprocher des fragments, reconnaître un signe, ordonner un groupe, segmenter un mot, choisir une valeur, restituer une lacune et comparer un parallèle. Chacun de ces choix peut être révisé lorsque paraît une meilleure photographie, lorsqu'un fragment est remonté, lorsqu'un nouveau parallèle est identifié ou lorsqu'une analyse grammaticale plus convaincante est proposée.

Cette évolution n'est pas une faiblesse de la discipline : elle est le fonctionnement normal de la recherche. Une même inscription peut être rééditée et réinterprétée pendant des décennies. Les éditions successives ne doivent pas être réduites à une opposition entre une ancienne erreur et une nouvelle vérité. Chacune témoigne d'un état des sources, des connaissances et du raisonnement.

Loracrafft est conçu pour conserver cette pluralité plutôt que pour imposer une réponse unique. Il distingue ce qui est matériellement observé, ce qui est proposé par un moteur, ce qui est validé par un chercheur et ce qui est retenu dans une édition. Il permet ainsi de répondre non seulement à la question « que dit ce texte ? », mais aussi à des questions proprement scientifiques : « sur quels signes cette lecture repose-t-elle ? », « quelles alternatives ont été envisagées ? », « quelle lacune a été restituée ? », « quel parallèle soutient ce choix ? » et « pourquoi la traduction aurait-elle changé depuis l'édition précédente ? ».

La rigueur de l'architecture informatique répond donc directement à une exigence philologique. La traçabilité protège la discussion savante ; le versionnement rend les révisions citables ; la séparation des niveaux empêche une hypothèse de devenir silencieusement un fait ; la conservation des désaccords permet à de nouvelles recherches de reprendre le dossier sans recommencer depuis zéro. Loracrafft ne cherche pas à figer l'interprétation, mais à donner à son évolution une mémoire scientifique.

1. Vision et objectifs du projet Loracrafft

1.1. Positionnement

Loracrafft n'est pas un simple OCR¹ de hiéroglyphes. Un OCR classique cherche principalement à associer une forme visuelle à un caractère. Une inscription égyptienne exige une chaîne de décisions beaucoup plus riche : identifier le support et la zone inscrite, restaurer une géométrie de lecture, distinguer un signe d'un fragment ou d'un bruit, ordonner les groupes dans les cadrats, proposer plusieurs identifications, déterminer leurs valeurs contextuelles, translittérer le texte, segmenter les mots, analyser la morphologie et la syntaxe, rechercher dans les corpus des textes ou passages parallèles et confronter leurs graphies, leurs formulations et leurs contextes, avant d'en proposer une traduction.

Loracrafft est donc conçu comme une **plateforme d'observation, d'interprétation et d'argumentation**. Il transforme progressivement une source visuelle en édition scientifique, sans confondre les données observées avec les hypothèses qui en sont tirées.

La valeur durable du projet ne réside pas dans un modèle de vision particulier. Les moteurs de Deep Learning², de reconnaissance par gabarits³ ou les services multimodaux⁴ sont remplaçables.

¹ OCR (Optical Character Recognition, ou reconnaissance optique de caractères) : procédé informatique permettant de convertir l'image d'un texte en caractères exploitables par un ordinateur. Appliqué aux hiéroglyphes, il doit également tenir compte de l'orientation des signes, de leur disposition spatiale, de leur regroupement en cadrats et de leur état de conservation. Loracrafft dépasse ainsi le cadre d'un OCR traditionnel.

² Deep learning, ou apprentissage profond : méthode d'intelligence artificielle utilisant des réseaux de neurones comportant plusieurs couches de traitement. Le système apprend, à partir d'un grand nombre d'exemples annotés, à repérer automatiquement les caractéristiques utiles à une tâche. Dans Loracrafft, cette méthode pourrait servir à détecter, segmenter ou identifier des signes hiéroglyphiques, sans que ses résultats constituent pour autant une validation scientifique.

³ Reconnaissance par gabarits : méthode consistant à comparer l'image d'un signe inconnu avec des images de référence, appelées gabarits, dont l'identité est déjà connue. Le système mesure leur ressemblance de forme et propose le ou les signes les plus proches. Cette méthode est simple et explicable, mais elle est sensible aux différences de taille, d'orientation, de style, d'usure et de qualité d'image. Une forte ressemblance avec un gabarit ne constitue donc pas, à elle seule, une identification scientifique.

⁴ Services multimodaux : services d'intelligence artificielle capables de traiter conjointement plusieurs formes d'information, par exemple une image et une instruction textuelle. Dans Loracrafft, ils peuvent examiner la photographie d'une inscription, proposer des délimitations, des identifications ou une transcription et expliquer leurs réponses. Leurs résultats restent toutefois des hypothèses : ils doivent

La valeur durable réside dans :

- le modèle de données commun ;
- la séparation entre faits, hypothèses et décisions ;
- la provenance de chaque transformation ;
- les connaissances égyptologiques structurées ;
- les mécanismes de révision et de comparaison ;
- la possibilité de reproduire et de contester une conclusion.

1.2. Objectifs scientifiques

Loracrafft poursuit six objectifs principaux.

Produire une chaîne interprétative explicable

Une traduction doit pouvoir être remontée jusqu'aux glyphes, aux zones de l'image et aux décisions intermédiaires dont elle dépend. L'utilisateur doit pouvoir demander : « pourquoi ce signe ? », « pourquoi cette segmentation ? », « quel parallèle⁵ justifie cette restitution ? » ou « que devient la traduction si l'hypothèse B est retenue ? ».

Préserver la pluralité des lectures

Les inscriptions lacunaires, atypiques ou mal photographiées ne conduisent pas toujours à une solution unique. Loracrafft se veut pouvoir conserver des branches concurrentes et éviter de réduire prématurément l'incertitude à une seule valeur.

Associer l'expertise humaine au calcul

Le système n'a pas vocation à remplacer l'égyptologue. Il doit accélérer les tâches répétitives, exposer les éléments pertinents et formaliser les raisonnements. Une validation humaine est une contribution scientifique historisée, jamais une correction silencieuse.

Constituer des corpus réutilisables

Chaque inscription validée peut enrichir des collections de formes, de graphies, de séquences, de constructions grammaticales et de traductions, sous réserve de droits et d'un niveau de confiance documenté.

être enregistrés avec la version du service utilisé, puis contrôlés et éventuellement corrigés par un spécialiste.

⁵ Parallèle : texte ou passage distinct présentant des ressemblances significatives avec le texte étudié ; mêmes signes, même expression, même construction grammaticale ou contenu comparable. Un parallèle peut aider à identifier un signe, restituer une lacune ou éclairer une traduction, mais il doit toujours être examiné dans son propre contexte historique et documentaire.

Permettre l'évaluation comparative

Les moteurs doivent pouvoir être remplacés et comparés sur les mêmes entrées. Les performances sont mesurées par étapes : détection, classification, reconnaissance, ordre de lecture, encodage MdC, analyse linguistique et traduction.

Favoriser l'interopérabilité

Les formats internes de Loracrafft doivent pouvoir être exportés vers des standards et des usages existants : les formats courants d'image avec leurs annotations ; le Manuel de Codage pour représenter les séquences hiéroglyphiques ; les caractères hiéroglyphiques Unicode lorsqu'un caractère approprié existe ; TEI/EpiDoc pour l'édition scientifique ; IIIF pour publier, consulter et annoter les images ; et JSON-LD pour sérialiser les métadonnées selon le modèle de provenance PROV-O⁶.

1.3. Objectifs fonctionnels

À terme, la plateforme doit permettre :

1. d'importer une ou plusieurs photographies d'une inscription ;
2. de documenter le support, le contexte, la localisation et les droits ;
3. de corriger la perspective, l'orientation, le contraste et le bruit ;
4. de détecter les zones, bandes, lignes, colonnes, cadrats et cartouches ;
5. de déterminer un ou plusieurs sens de lecture plausibles ;
6. d'extraire et de classer les objets visuels ;
7. de proposer des identifications Gardiner sans jamais inventer un signe ;
8. de recueillir et d'historiser les validations humaines ;
9. de produire une représentation MdC reliée aux objets visuels ;
10. d'analyser la translittération, la morphologie et la syntaxe ;

⁶ La TEI (Text Encoding Initiative) fournit des règles pour représenter numériquement des textes et leur structure. EpiDoc est une adaptation de la TEI destinée aux inscriptions et aux documents anciens ; elle permet notamment de décrire les lacunes, les restitutions, les lectures incertaines, les corrections et les variantes éditoriales. IIIF (International Image Interoperability Framework) est un ensemble de normes facilitant la diffusion d'images scientifiques en haute définition. Il permet de zoomer, d'afficher une région précise, de comparer plusieurs images et d'y associer des annotations sans dupliquer les fichiers originaux. JSON-LD (JavaScript Object Notation for Linked Data) est une forme de JSON destinée aux données liées. Elle permet d'associer les champs d'un document à des identifiants et à des vocabulaires partagés afin que des logiciels différents puissent les interpréter de manière cohérente. PROV-O est un modèle de provenance normalisé par le W3C. Il permet de décrire les objets utilisés ou produits, les opérations qui les ont transformés et les personnes ou logiciels qui en sont responsables. JSON-LD peut être utilisé pour écrire des données organisées selon ce modèle.

11. de rechercher des parallèles dans des corpus documentés ;
12. de générer des traductions argumentées et révisables ;
13. de produire des textes audio à écouter sur un smartphone ou une tablette
14. de publier un dossier scientifique complet et exportable (option).

1.4. Non-objectifs

Loracrafft ne doit pas :

- masquer l'incertitude derrière une sortie apparemment certaine ;
- remplacer une lacune par un signe plausible sans marquage explicite ;
- confondre proximité visuelle, identité paléographique et valeur linguistique ;
- utiliser automatiquement une correction humaine comme vérité universelle ;
- produire une traduction sans conserver ses dépendances ;
- modifier une observation ancienne pour la faire correspondre à une conclusion nouvelle ;
- dépendre définitivement d'un fournisseur, d'un modèle ou d'un format propriétaire.

1.5. Utilisateurs

Les principaux profils sont :

- **chercheur ou épigraphiste**, qui contrôle les hypothèses et produit l'édition ;
- **contributeur**, qui annote des images ou valide des signes dans un périmètre défini ;
- **responsable de corpus**, qui organise les sources, les droits et les versions ;
- **ingénieur**, qui développe ou évalue un moteur ;
- **lecteur**, qui consulte une édition, ses preuves et ses variantes ;
- **administrateur scientifique**, qui gère vocabulaires, référentiels et politiques de validation.
- **visiteur d'un site, d'un monument ou d'un musée**, qui utiliserait Loracrafft pour identifier et comprendre une inscription *in situ* et pourra, lorsque les droits le permettent, transmettre une photographie ou signaler un nouveau témoin.

2. Principes scientifiques

2.1. Principe premier

Toute conclusion doit pouvoir être expliquée, reproduite et contestée.

Ce principe entraîne quatre obligations :

- les observations sont immuables ;
- les hypothèses sont séparées des faits ;
- les validations et réfutations sont historisées ;
- chaque décision possède une provenance et des preuves.

2.2. Séparation des niveaux épistémiques

Loracrafft distingue au minimum :

1. **source** : photographie, relevé, fac-similé ou édition importée ;
2. **observation** : contour, couleur, orientation, position, relation spatiale ;
3. **hypothèse** : classification ou identification proposée ;
4. **assertion** : proposition interprétative explicite ;
5. **validation** : acte par lequel un agent accepte, rejette ou nuance une hypothèse ;
6. **conclusion** : résultat argumenté dépendant d'un ensemble d'assertions ;
7. **publication** : sélection éditoriale figée dans une version.

Une observation n'est jamais formulée comme une conclusion. Par exemple, « objet sombre de 42 × 18 pixels » est une observation ; « signe N35 » est une hypothèse ; « préposition *m* » est une interprétation linguistique ; « dans » est un choix de traduction.

2.3. Immutabilité et enrichissement

Chaque module ajoute une nouvelle couche au LoraDocument. Il ne réécrit pas la couche produite par un module antérieur. Une rectification crée un nouvel objet ou un nouvel événement qui :

- référence l'objet rectifié ;
- expose l'auteur et la date ;
- donne la justification ;
- indique si l'objet précédent est remplacé pour un usage donné ;
- conserve l'état antérieur consultable.

Les fichiers sources sont identifiés par une empreinte cryptographique. Les dérivés sont eux-mêmes versionnés et reliés à leurs paramètres de production.

2.4. Honnêteté face à l'absence de résultat

L'absence de candidat est un résultat acceptable. Les états UNKNOWN, NEEDS_REVIEW, UNREADABLE ou NOT_APPLICABLE sont préférables à une identification fabriquée. Un moteur doit publier ses limites : domaine d'entraînement, familles de supports, époques, résolutions, métriques et seuils⁷.

2.5. Réversibilité

Une étape interprétative ne doit pas rendre impossible le retour aux alternatives. Les interfaces permettent de recalculer les conséquences d'une nouvelle validation sans recommencer les traitements indépendants. Les sorties dérivées doivent déclarer exactement les entrées dont elles dépendent.

2.6. Identifiants stables

Tout objet scientifique⁸ reçoit un identifiant unique et stable. Les identifiants ne doivent pas dépendre de l'ordre dans un tableau JSON⁹. Les identifiants locaux peuvent être des UUID¹⁰ ; les objets publiés peuvent recevoir des URI¹¹ pérennes.

Les coordonnées peuvent évoluer avec les transformations d'image, mais l'identité d'un objet est maintenue par des relations explicites telles que `derived_from`, `replaces`, `same_as` ou `fragment_of`.

⁷ Un seuil est une valeur numérique à partir de laquelle le moteur applique une décision à un résultat. Par exemple, il peut proposer automatiquement une identification lorsque son score dépasse 0,80% et demander une vérification humaine en dessous de cette valeur. Le seuil dépend du moteur, du corpus et de la tâche considérée : il doit être justifié et documenté, et ne constitue jamais à lui seul une validation scientifique.

⁸ Dans Loracrafft, un objet scientifique est une information individualisée et conservée par le système : image source, zone inscrite, registre, cadrat, signe détecté, hypothèse de lecture, validation humaine, translittération ou traduction.

⁹ JSON (JavaScript Object Notation) est un format textuel utilisé pour enregistrer et échanger des données structurées. Un tableau JSON est simplement une liste ordonnée d'objets. La position d'un objet dans cette liste peut changer et ne constitue donc pas un identifiant fiable.

¹⁰ Un UUID (Universally Unique Identifier, ou identifiant universellement unique) est un code généré automatiquement, par exemple 550e8400-e29b-41d4-a716-446655440000. La probabilité que deux objets reçoivent accidentellement le même UUID est négligeable.

¹¹ Une URI (Uniform Resource Identifier) est un nom ou une adresse permettant d'identifier durablement une ressource publiée. Elle est dite pérenne lorsqu'elle reste valable malgré un changement de serveur, de logiciel ou d'organisation interne. Les DOI et les identifiants ARK sont des exemples courants d'identifiants pérennes.

2.7. Pluralité et contradiction

Deux hypothèses incompatibles peuvent coexister. La contradiction est modélisée, non supprimée. Une politique de sélection peut désigner une hypothèse préférée pour une édition donnée, sans effacer les alternatives.

2.8. Reproductibilité

Toute activité automatique enregistre :

- le module et sa version ;
- le moteur et sa version ;
- l’empreinte du modèle ou du catalogue ;
- les paramètres ;
- les identifiants des entrées ;
- l’environnement utile à la reproduction ;
- les heures de début et de fin ;
- les avertissements et erreurs ;
- les identifiants des sorties.

2.9. Validation humaine qualifiée

Une validation humaine n’a de sens qu’avec son contexte. Le système conserve le rôle du validateur, son domaine de compétence déclaré, l’interface utilisée, les éléments visibles au moment de la décision et le niveau de certitude choisi.

Le consensus peut être calculé, mais ne remplace pas l’argument scientifique.

2.10. Évaluation par niveau

La qualité globale ne doit pas être résumée par un score unique. Les métriques possibles comprennent :

- intersection sur union pour les régions ;
- précision, rappel et F1¹² pour la détection ;
- taux de réussite top-1 et top-k¹³ pour Gardiner ;

¹² La précision indique, parmi les signes détectés par le moteur, la proportion de détections correctes. Le rappel indique, parmi les signes réellement présents dans l’image, la proportion effectivement retrouvée. Le score F1 combine ces deux mesures : il n’est élevé que si le moteur produit peu de fausses détections tout en oubliant peu de signes. Sa valeur varie de 0, pour un échec complet, à 1, pour un résultat parfait.

¹³ Pour chaque signe, le moteur classe plusieurs codes Gardiner du plus probable au moins probable. Le taux de réussite top-1 mesure la proportion de cas où le code correct arrive en première position. Le taux top-k mesure la proportion de cas où il

- taux d'erreur de séquence pour l'ordre de lecture ;
- accord inter-annotateurs ;
- exactitude de segmentation lexicale ;
- scores morphosyntaxiques ;
- qualité de récupération des parallèles ;
- évaluation humaine des traductions.

figure parmi les k premières propositions — par exemple parmi les cinq premières pour un top-5. Cette seconde mesure évalue l'utilité du moteur comme assistant proposant une liste de candidats, mais la présence du bon code dans cette liste ne constitue pas une validation scientifique.

3. Architecture générale

3.1. Vue en couches

L'architecture logique comprend six couches :

1. **sources** : fichiers, métadonnées et droits ;
2. **observations visuelles** : images dérivées et géométrie ;
3. **hypothèses glyphiques** : objets, candidats et validations ;
4. **représentations textuelles** : ordre de lecture, MdC et translittération ;
5. **interprétations linguistiques** : lexique, morphologie, syntaxe et parallèles ;
6. **conclusions éditoriales** : raisonnement, traduction et publication.

LoraWorkflow orchestre les traitements. Le LoraDocument sert de dossier scientifique partagé et de journal des dépendances.

LoraCapture



LoraImage → LoraLayout → LoraDirection → LoraGlyph → LoraReview



LoraMdC → LoraGrammar ↔ LoraLexicon



LoraCorpus → LoraReasoner



LoraTranslate → LoraSpeech

LoraWorkflow orchestre tous les modules.

Chaque module lit certaines collections et écrit une nouvelle couche.

3.2. Architecture modulaire

Chaque module expose :

- un contrat d'entrée ;
- un contrat de sortie ;
- un manifeste de capacités ;
- une politique d'erreur ;
- des événements de progression ;
- une description de provenance ;
- une suite de tests de conformité.

Le cœur métier ne dépend pas de l'interface en ligne de commande, de l'interface graphique ni du moteur particulier. L'implémentation actuelle de LoraGlyph suit déjà ce principe : service.py orchestre un extracteur, un classificateur et un GlyphRecognizer remplaçable.

3.3. Collections logiques

Le document commun comprend notamment :

- `sources` ;
- `images` ;
- `layout_objects` ;
- `reading_orders` ;
- `glyph_hypotheses` ;
- `human_validations` ;
- `mdc_encodings` ;
- `linguistic_analyses` ;
- `lexical_links` ;
- `corpus_matches` ;
- `reasoning_objects` ;
- `translations` ;
- `speech_outputs` ;
- `activities` ;
- `history`.

Ces collections sont logiques : elles peuvent être stockées dans un document JSON, plusieurs fichiers adressés par contenu ou une base de données. Leur sémantique demeure stable.

3.4. Stockage

Une architecture hybride est recommandée :

- stockage objet pour les images et artefacts volumineux ;
- base documentaire pour les LoraDocument et projections JSON ;
- index de recherche pour les corpus et les séquences ;
- graphe de provenance pour les dépendances complexes ;
- journal append-only pour les événements et validations.

Le stockage physique ne doit pas devenir le contrat public. Un dépôt local sur disque doit rester possible pour le travail hors ligne.

3.5. Événements et orchestration

Les modules peuvent être appelés directement ou par événements. Exemples :

- `image.normalized` déclenche une analyse de mise en page ;
- `layout.reviewed` autorise le calcul du sens de lecture ;
- `glyph.validation.added` invalide les MdC dépendants ;
- `mdc.selected` relance l'analyse grammaticale ;

- `corpus.index.updated` marque les anciens rapprochements comme recalculables.

L'invalidation signifie « sortie à recalculer », non « sortie à supprimer ».

3.6. Sécurité et droits

Les sources peuvent être inédites ou soumises à restrictions. Le contrôle d'accès s'applique au niveau du document, de l'artefact et, si nécessaire, de la région. Les exports héritent de la politique la plus restrictive de leurs dépendances, sauf décision explicite d'un responsable habilité.

4. Modèle de données : LoraDocument

4.1. Rôle

Le LoraDocument est l'unité scientifique centrale. Il ne représente pas seulement un texte final : il contient l'historique structuré de la transformation d'une ou plusieurs sources en interprétations.

Un document peut correspondre à une inscription complète, un fragment, une face de monument ou un dossier multi-images. Les relations entre documents permettent de représenter remontages, duplicata, parallèles ou publications successives.

4.2. Enveloppe minimale

```
{
  "$schema": "https://loracrafft.org/schemas/loradocument/1.0",
  "document_id": "urn:lora:document:01K0...",
  "revision_id": "urn:lora:revision:01K0...",
  "created_at": "2026-07-29T10:00:00Z",
  "updated_at": "2026-07-29T10:12:00Z",
  "title": "Stèle, face A, registre supérieur",
  "status": "working",
  "sources": [],
  "images": [],
  "layout_objects": [],
  "reading_orders": [],
  "glyph_hypotheses": [],
  "human_validations": [],
  "mdc_encodings": [],
  "linguistic_analyses": [],
  "lexical_links": [],
  "corpus_matches": [],
  "reasoning_objects": [],
  "translations": [],
  "speech_outputs": [],
  "activities": [],
  "history": []
}
```

4.3. Types communs

Tout objet métier partage un noyau :

```
{
  "id": "urn:lora:glyph-hypothesis:...",
  "type": "GlyphHypothesis",
  "created_at": "2026-07-29T10:04:00Z",
  "created_by": {"kind": "software", "id": "loraglyph"},
  "activity_id": "urn:lora:activity:...",
  "source_refs": ["urn:lora:layout-object:..."],
  "status": "proposed",
  "tags": []
}
```

Les objets géométriques utilisent un système de coordonnées déclaré :

```
{
  "geometry": {
    "space": "urn:lora:image:normalized-1",
    "shape": "polygon",
    "points": [[46, 18], [84, 18], [84, 55], [46, 55]]
  }
}
```

Un simple bbox reste accepté pour les prototypes, mais un polygone est préférable sur les inscriptions inclinées, courbes ou irrégulières.

4.4. Sources et images

Une Source décrit le témoin documentaire : fichier, photographie, dessin ou référence bibliographique. Une ImageArtifact décrit une représentation numérique et ses transformations.

Champs recommandés :

- empreinte SHA-256 ;
- type MIME, dimensions et profil colorimétrique ;
- auteur de la prise de vue et date ;
- institution, cote et contexte archéologique ;
- licence et restrictions ;
- relation à la source physique ;
- chaîne de transformations.

4.5. Objets de mise en page

Un LayoutObject représente une région observée : surface inscrite, registre, bande, colonne, ligne, cadrat, cartouche, séparateur, lacune ou objet indéterminé. Les relations spatiales sont explicites : contains, precedes, overlaps, aligned_with, part_of.

4.6. Hypothèses glyphiques

Une hypothèse associe un objet visuel à une classification et à zéro ou plusieurs candidats. Les scores sont propres à un moteur ; ils ne sont pas assimilés automatiquement à des probabilités scientifiques.

```
{
  "id": "urn:lora:glyph-hypothesis:gh-42",
  "type": "GlyphHypothesis",
  "layout_object_id": "urn:lora:layout-object:lo-42",
  "classification": "GLYPH",
  "candidates": [
    {
      "sign_ref": "gardiner:G17",
      "score": 0.87,
    }
  ]
}
```

```

    "score_kind": "engine_confidence",
    "engine": "template"
  },
  {
    "sign_ref": "gardiner:G43",
    "score": 0.08,
    "score_kind": "engine_confidence",
    "engine": "template"
  }
],
"status": "needs_review"
}

```

4.7. Validations

Une validation est un événement autonome. Elle cible une assertion, indique une action (accept, reject, amend, defer) et peut proposer une nouvelle valeur.

```

{
  "id": "urn:lora:validation:v-17",
  "target_id": "urn:lora:glyph-hypothesis:gh-42",
  "action": "amend",
  "selected_value": "gardiner:G17",
  "certainty": "high",
  "agent": {"kind": "human", "id": "urn:lora:person:p-9"},
  "rationale": "Forme de la chouette et parallèle dans la colonne voisine.",
  "evidence_refs": ["urn:lora:image-region:...", "urn:lora:match:m-8"],
  "created_at": "2026-07-29T10:08:00Z"
}

```

4.8. Représentations textuelles

Une représentation MdC est une vue ordonnée des signes et groupes. Elle doit conserver un alignement bidirectionnel entre les segments de chaîne et les objets glyphiques. La translittération, la segmentation lexicale et l'analyse grammaticale sont des objets distincts, afin qu'une même séquence graphique puisse recevoir plusieurs analyses.

4.9. Conclusions et traductions

Une traduction référence :

- la branche d'analyse retenue ;
- les segments sources ;
- la langue cible et le registre ;
- le degré de littéralité ;
- les choix éditoriaux ;
- les alternatives écartées ;
- l'auteur ou le moteur ;
- les justifications.

4.10. Révisions

Une révision est un instantané logique d'un document. Elle ne duplique pas nécessairement tous les objets physiques. Elle pointe vers un ensemble cohérent d'identifiants et fournit une empreinte du manifeste. Une publication est une révision signée ou figée, assortie d'un statut et éventuellement d'un DOI ou d'une URI pérenne.

4.11. État de l'implémentation au 17 août 2026

Le noyau LoraDocument 1.0 est désormais implémenté dans le prototype Python. Il comprend :

- création et validation de l'enveloppe minimale ;
- schéma JSON <https://loracraftt.org/schemas/loradocument/1.0> ;
- identifiants stables indépendants de l'ordre des régions importées ;
- adaptateur du JSON historique LoraLayout vers layout_objects ;
- adaptateur du JSON historique LoraGlyph vers des glyph_hypotheses séparées ;
- création de human_validations autonomes sans modification de l'hypothèse ciblée ;
- coexistence de validations contradictoires provenant de plusieurs annotateurs ;
- journal JSONL append-only ;
- écriture JSON atomique et contrôle des références ;
- provenance de l'image originale, de l'image redressée, de l'image contrastée et des activités automatiques.

Cinq tests de fond valident actuellement l'import non destructif, la stabilité des identifiants, la séparation des couches, la coexistence de deux annotateurs et le journal append-only. Ces tests sont des tests de contrat ; ils ne mesurent pas encore la qualité scientifique de la segmentation ou de la reconnaissance.

5. Description détaillée de chaque module

5.1. LoraCapture

Mission

LoraCapture acquiert les sources visuelles et documentaires. Son rôle est de préserver l'original, de collecter les métadonnées et de créer un point de départ fiable. Il ne corrige ni n'interprète l'image.

Entrées et sorties

Les entrées peuvent être des fichiers locaux, des images IIF, des lots photographiques, des scans, des relevés ou des imports depuis un catalogue. La sortie comprend une Source, un ou plusieurs ImageArtifact originaux et une activité d'import.

Le module calcule une empreinte, détecte le type de fichier, relève les dimensions, conserve les métadonnées EXIF pertinentes et vérifie l'intégrité. Il permet d'associer plusieurs vues : lumière rasante, infrarouge, photogrammétrie, détail et vue d'ensemble.

Contrôles scientifiques

Le module doit empêcher la confusion entre :

- objet physique et fichier ;
- original et copie de travail ;
- photographie et fac-similé ;
- absence de métadonnées et valeur négative.

La description contextuelle inclut, lorsque disponible, institution, cote, provenance, datation proposée, matériau, dimensions, emplacement sur le monument et bibliographie. Ces informations sont elles-mêmes sourcées et peuvent être hypothétiques.

Erreurs et qualité

Les fichiers illisibles, doublons exacts, profils colorimétriques absents et métadonnées incohérentes produisent des avertissements. Un doublon n'est pas supprimé automatiquement : il est relié par `same_content_as`.

Contrat, invariants et validation

L'opération d'import reçoit un URI ou un flux binaire, les métadonnées disponibles et l'identité du déposant. Elle retourne un manifeste de source et les références d'artefacts ; elle ne retourne jamais une image modifiée sous l'identité de l'original. Deux imports des mêmes octets peuvent créer deux événements de dépôt, mais doivent converger vers le même contenu adressé par empreinte. Les noms de fichiers, dates EXIF et extensions ne sont pas considérés comme des identifiants fiables.

Les contrôles portent sur l'intégrité binaire, la cohérence MIME, les dimensions, la lisibilité, les droits et la complétude minimale du contexte. Un jeu d'essai doit inclure des doublons, des fichiers tronqués, des métadonnées contradictoires, plusieurs vues d'un objet et des images sans EXIF. Le critère d'acceptation scientifique est qu'un tiers puisse identifier exactement le fichier utilisé, ses conditions d'accès et son rapport avec l'objet documenté.

État et évolution

Ce module relève de l'architecture cible. Une première version peut se limiter à l'import local, au calcul SHA-256, à un manifeste JSON et à la gestion des licences.

5.2. LoraImage

Mission

LoraImage produit des dérivés adaptés à l'analyse tout en conservant l'image originale. Il prend en charge rotation, redressement, correction de perspective, contraste, débruitage, recadrage et éventuellement assemblage.

Principe de transformation

Chaque transformation est non destructive et fournit :

- l'identifiant de l'image source ;
- la matrice ou la fonction géométrique ;
- les paramètres photométriques ;
- une image dérivée ;
- une mesure ou une appréciation de qualité ;
- la transformation inverse lorsqu'elle est définie.

La conservation des transformations permet de reporter une région détectée sur l'image normalisée vers l'original. C'est essentiel pour l'audit scientifique et l'affichage comparatif.

Méthodes

La correction de perspective peut recevoir quatre coins validés ou proposer des coins candidats. Le redressement estime l'écart aux axes dominants sans supposer que le texte est horizontal. Le contraste local de type CLAHE et le débruitage doivent rester paramétrables. Une binarisation peut servir à un moteur sans devenir une nouvelle « vérité » sur l'image.

Contrôle humain

Une prévisualisation avant/après et une superposition sont recommandées. L'utilisateur peut accepter, ajuster ou rejeter une transformation. Les réglages manuels sont enregistrés comme paramètres d'activité.

Contrat, invariants et validation

L'entrée est un ImageArtifact identifié ; la sortie est un nouvel artefact, une transformation géométrique et un rapport de qualité. Les coordonnées ne sont jamais converties implicitement : chaque géométrie déclare son espace image. Lorsqu'une opération n'est pas inversible — seuillage, compression destructive, suppression de fond — le module le signale et conserve le lien vers l'original.

Les tests vérifient l'absence de recadrage involontaire, la stabilité des dimensions déclarées, l'aller-retour des points par la matrice inverse et la conservation des couleurs lorsqu'aucune correction photométrique n'est demandée. Des mires et images annotées permettent de mesurer l'erreur de redressement. Sur une image réelle, l'acceptation dépend aussi de la lisibilité : une amélioration d'un score de contraste ne justifie pas la disparition de traits fins. Plusieurs profils de préparation peuvent coexister pour des moteurs différents.

État actuel

Le prototype conserve désormais l'original, l'image redressée et l'image contrastée comme artefacts distincts. Le redressement utilise des segments détectés par transformée de Hough, puis calcule une orientation dominante avec une moyenne circulaire modulo 90° afin de ne pas confondre colonnes verticales et inclinaison du document.

Sur la stèle réelle utilisée pour les essais, l'inclinaison estimée était d'environ $44,7^\circ$. Le premier pipeline exécutait la segmentation avant un redressement correct et produisait donc une géométrie inutilisable. La correction du sens de rotation et le déplacement du redressement avant LoraLayout ont validé le principe. L'angle détecté et l'angle appliqué sont maintenant des paramètres de provenance.

La détection automatique des quatre coins et la correction de perspective entièrement automatique ne sont toujours pas réalisées. Le résultat du redressement doit rester contrôlable visuellement : une orientation mathématiquement dominante peut appartenir au décor, à un bord cassé ou à un élément moderne plutôt qu'au texte.

5.3. LoraLayout

Mission

LoraLayout décrit exclusivement la géométrie de l'inscription. Il détecte des régions et leurs relations sans attribuer de valeur hiéroglyphique.

Objets

Le vocabulaire minimal comprend :

- INSCRIPTION_SURFACE ;
- REGISTER ;
- BAND ;
- ROW ;

- COLUMN ;
- QUADRAT ;
- CARTOUCHE ;
- SEPARATOR ;
- DAMAGE ;
- UNRESOLVED_REGION.

Un objet possède une géométrie, un type proposé, une confiance technique, une origine et des relations. Plusieurs découpages concurrents peuvent être conservés.

Algorithmes

Les premiers moteurs peuvent utiliser projections, contours, morphologie et heuristiques de taille. Des détecteurs entraînés pourront ensuite produire le même contrat. Le module doit gérer inscriptions horizontales, verticales, mixtes, rétrogrades, circulaires ou organisées en scènes.

Cartouches et lacunes

Un contour allongé n'est pas nécessairement un cartouche. LoraLayout propose une région de type cartouche ; la validation peut ensuite mobiliser contenu, contexte et corpus. Les contours ouverts ou coupés par le bord doivent recevoir un statut spécifique, non être ignorés.

Contrat, invariants et validation

Le module reçoit une image de travail et, éventuellement, une surface inscrite présélectionnée. Il retourne des objets géométriques, leurs relations et les diagnostics du moteur. Aucun objet ne reçoit de code Gardiner ou de valeur linguistique. Les régions peuvent se chevaucher lorsqu'elles expriment des analyses concurrentes ; un arbre de contenance validé ne doit toutefois contenir ni cycle ni référence à un espace image incompatible.

L'évaluation sépare détection des régions, typage et relations. Les jeux de test doivent couvrir texte horizontal, colonnes, registres, cartouches ouverts, signes jointifs, relief très faible et arrière-plans décorés. L'interface de revue affiche faux positifs et régions manquées, et permet de scinder, fusionner ou redessiner un polygone. Le rapport final donne précision et rappel par type, distribution des tailles et résultats par famille de support, afin qu'une bonne moyenne ne masque pas l'échec sur les cas difficiles.

État actuel

L'implémentation expérimentale détecte orientation générale, bandes, cadrats et cartouches par heuristiques. Elle a été corrigée pour les colonnes verticales, le filtrage des séparateurs et la déduplication de cartouches.

Les essais sur images réelles ont toutefois invalidé l'hypothèse implicite « une composante connexe égale un signe ». Une première fusion a réduit 173 composantes brutes à 56 groupes candidats, mais elle restait simultanément trop conservatrice pour certains signes fragmentés et trop agressive pour des voisins proches. Des régions

étrangères au texte — par exemple un animal ou un motif figuratif — ont également montré la nécessité des classes `NON_GLYPH` ou `OUTLIER`, sans forcer leur passage dans Gardiner.

L'échec d'une version sur une inscription à deux lignes, qui ne produisait aucun cadrat attendu, a conduit à inverser la stratégie : détecter d'abord surface, registres, lignes, colonnes et séparateurs, puis rechercher les cadrats dans chaque structure. Cette stratégie **register-first** est devenue un invariant architectural. Sur `test_2`, les deux registres sont désormais correctement séparés, mais les vingt cadres cyan produits lors d'un essai ne correspondent pas encore tous aux véritables unités hiéroglyphiques. Le découpage reste donc non validé.

Les cartouches exigent un pipeline spécialisé : fermeture morphologique, contours ouverts, traitement des régions coupées par le bord et validation par le contenu. Les améliorations expérimentales réduisent certains faux positifs, mais ne constituent pas encore une détection scientifique validée.

5.4. LoraDirection

Mission

LoraDirection établit l'ordre de lecture probable à partir de la géométrie, de signes orientables et de conventions de composition. Il ne modifie pas les objets de mise en page : il produit un ou plusieurs graphes d'ordre.

Représentation

Un `ReadingOrder` comprend :

- la direction générale : droite-vers-gauche, gauche-vers-droite, haut-vers-bas ;
- l'ordre des bandes ;
- l'ordre interne aux cadrats ;
- les ambiguïtés ;
- les indices utilisés ;
- un score et un statut de validation.

Un graphe partiel est préférable à une liste forcée lorsque certaines relations sont inconnues.

Indices

Le module combine :

- orientation de figures humaines, animales ou divines ;
- alignement des colonnes et lignes ;
- position des titres et formules ;
- symétries de scène ;
- signes de fermeture ou de début connus ;
- éventuel parallèle de corpus.

Chaque indice reste accessible. Une majorité de signes tournés vers la droite ne suffit pas toujours : scènes symétriques et inscriptions rétrogrades exigent une branche alternative.

Interaction

Si les signes orientables ne sont pas encore identifiés, le module peut demander à LoraGlyph une classe d'orientation sans exiger un code Gardiner définitif. Après validation de signes, un nouvel ordre peut être calculé.

Contrat, invariants et validation

L'entrée comprend objets de mise en page, relations spatiales et indices d'orientation. La sortie est un graphe orienté dont les arêtes signifient « doit précéder » ou « précède probablement ». Une linéarisation n'est produite que pour une branche sans cycle. Si les indices créent un cycle, le diagnostic cite les arêtes responsables et demande une revue au lieu de supprimer arbitrairement une contrainte.

Les tests utilisent des compositions horizontales, verticales, mixtes, boustrophédon hypothétique, symétries et inversions honorifiques. Les métriques portent sur les relations par paires et sur la séquence finale, car deux ordres peuvent différer localement tout en partageant une grande partie de leur structure. L'acceptation exige aussi une explication lisible : signes orientables observés, conventions appliquées, poids relatif des indices et raisons du maintien d'une alternative.

État

Le composant n'est pas encore implémenté dans le prototype actuel.

5.5. LoraGlyph

Mission

LoraGlyph extrait les objets visuels et produit des hypothèses d'identification. Il classe chaque objet comme GLYPH, GROUP, FRAGMENT, NOISE ou UNKNOWN. Les essais réels ont en outre fait apparaître le besoin de distinguer un objet figuratif valide mais étranger au répertoire (NON_GLYPH) d'un cas hors domaine statistique (OUTLIER). Ces deux valeurs sont candidates pour une évolution de schéma ; dans LoraDocument 1.0, elles doivent être conservées dans un diagnostic explicite associé à UNKNOWN, jamais perdues silencieusement. Le module ne produit ni MdC, ni translittération, ni traduction.

Pipeline

Le pipeline comprend :

1. extraction d'une vignette bornée avec marge ;
2. calcul de caractéristiques et classification primaire ;
3. reconnaissance pour les objets éligibles ;
4. classement de candidats ;
5. sauvegarde des vignettes et détails techniques ;
6. émission d'hypothèses et d'une activité.

FRAGMENT, GROUP et NOISE ne sont pas envoyés aveuglément au reconnaisseur. Ils peuvent faire l'objet de pipelines spécialisés : fusion de fragments, segmentation d'un groupe ou requalification.

Moteurs

L'interface `GlyphRecognizer` permet des moteurs par gabarits, CNN, Vision Transformer, recherche de voisins ou ensemble de modèles. Chaque candidat expose le référentiel, le code, le score, le type de score et le moteur.

Un ensemble ne doit pas masquer le désaccord entre moteurs. Il conserve les sorties élémentaires et décrit la règle de fusion.

Non-invention

Si aucun catalogue n'est disponible ou si les scores sont insuffisants, candidates reste vide et le statut devient `needs_review`. Un code ne peut être proposé que s'il appartient au référentiel déclaré ou à une extension explicitement nommée.

Contrat, invariants et validation

Le contrat reçoit une image, des objets et une configuration de moteurs. Il produit, pour chaque objet, exactement un résultat technique, y compris en cas d'échec. Les listes de candidats sont ordonnées, limitées à `top_k` et accompagnées de la version du référentiel. Un score n'est comparable entre deux moteurs que si une méthode de calibration commune est déclarée.

Les tests unitaires couvrent extraction aux bords, classification, absence de candidat, erreurs par objet et sérialisation. L'évaluation scientifique utilise des partitions sans fuite entre variantes photographiques. Elle publie `top-1`, `top-k`, matrice de confusion par catégorie Gardiner, couverture et taux de rejet honnête. Une campagne doit mesurer séparément signes isolés, groupes, fragments et supports dégradés. Les vignettes validées peuvent devenir un dataset uniquement après une étape de curation ; l'identifiant de l'annotation et la révision du crop restent alors attachés à l'exemple.

État actuel

`LoraGlyph 1.0.1` fournit toujours l'interface de moteur remplaçable, la classification primaire, les vignettes, les journaux et la reprise des corrections. Son ancien enrichissement du JSON reste disponible comme projection de compatibilité, mais le workflow actuel migre les résultats dans des `GlyphHypothesis` séparées du `LayoutObject`.

La base locale décompressée contient 6 722 fichiers glyphiques issus des ressources Raymond Monfort. Ce nombre désigne des fichiers ou variantes graphiques, non 6 722 signes Gardiner distincts. Le catalogue doit produire un manifeste indiquant au minimum code, variante, fichier, format, dimensions, empreinte, source, licence et éventuelle translittération. L'estimation antérieure d'environ 700 signes de base ne doit pas être confondue avec le nombre de formes disponibles.

La comparaison par gabarits a révélé sa fragilité aux différences de cadrage, d'épaisseur, de bruit et de segmentation. Un essai sur G1 a fourni un score d'environ 0,748 malgré un cas visuellement évident, ce qui interdit d'utiliser un seuil global non calibré comme décision scientifique.

Les sept invariants de Hu ont été introduits comme descripteur morphologique invariant à la translation, à l'échelle et, selon la transformation utilisée, à la rotation. Ils exigent cependant une binarisation cohérente, un contour normalisé et une gestion correcte des composantes multiples. Ils ne distinguent pas toujours des signes proches et ne résolvent aucune erreur de segmentation. Le moteur cible doit donc pouvoir combiner moments de Hu, recouvrement de silhouettes, profils de projection, descripteurs de contour et, ultérieurement, un CNN ou un Vision Transformer. Les scores élémentaires restent conservés séparément.

L'entraînement d'un CNN n'est pas prioritaire tant que la vérité terrain de segmentation et les partitions de corpus ne sont pas stabilisées. Un modèle plus puissant entraîné sur des cadrages incorrects ne ferait qu'apprendre les erreurs du pipeline.

5.6. LoraReview

Mission

LoraReview fournit les mécanismes de validation humaine. Il présente les preuves pertinentes, recueille une action et crée un événement sans écraser l'hypothèse.

Unité de revue

Une tâche de revue comprend :

- l'objet cible ;
- l'image originale et la vignette ;
- le contexte spatial ;
- les candidats et leurs moteurs ;
- les décisions antérieures ;
- les règles ou parallèles disponibles ;
- les actions autorisées.

L'interface doit permettre accepter, rejeter, corriger, déclarer illisible, fusionner, scinder ou reporter.

Consensus

Les politiques possibles sont : validateur unique, double lecture aveugle, majorité qualifiée ou arbitrage expert. La politique appartient au projet ou corpus, non au moteur. En cas de désaccord, les avis sont conservés et une décision éditoriale peut les départager.

Réutilisation

Une correction ne doit pas devenir automatiquement un exemple d'entraînement. Le passage au corpus d'apprentissage exige droits compatibles, contrôle qualité, séparation des jeux d'entraînement et de test, et traçabilité de la version de dataset.

Contrat, invariants et validation

La création d'une validation exige une cible existante, une action, un agent et un horodatage serveur. Une correction de valeur contient la nouvelle valeur et ne modifie pas l'hypothèse ciblée. L'interface enregistre également la révision et les preuves visibles afin de détecter un éventuel biais de présentation. Les actions sensibles, comme l'arbitrage ou la publication, sont réservées à des rôles.

La qualité de la revue se mesure par temps de décision, taux de report, accord inter-annotateurs, cohérence intra-annotateur et fréquence des arbitrages. Des tâches sentinelles peuvent détecter une dérive, mais elles ne doivent pas transformer l'expertise en simple classement. L'acceptation fonctionnelle exige navigation au clavier, zoom sans perte du contexte, reprise après interruption, absence d'écrasement en concurrence et export complet du journal. Toute décision doit pouvoir être annulée par une nouvelle décision, jamais par suppression.

État actuel

La validation console historique de LoraGlyph enregistre les corrections dans un journal JSONL append-only. Le noyau LoraDocument 1.0 généralise maintenant le principe : chaque décision humaine est un objet autonome avec cible, action, agent, certitude, justification et date. Deux annotateurs peuvent accepter, rejeter ou amender la même hypothèse sans s'écraser.

L'interface visuelle reste à développer. Elle devra prioritairement permettre de corriger la géométrie de test_2 — fusion, scission, suppression d'un faux objet et ajout d'une région manquante — avant la validation Gardiner. La revue de segmentation précède donc la revue de reconnaissance.

5.7. LoraMdC

Mission

LoraMdC transforme une branche de glyphes et un ordre de lecture validé ou proposé en Manuel de Codage. Il maintient l'alignement avec les objets visuels.

Construction

Le module produit une structure abstraite avant de sérialiser une chaîne. Les nœuds représentent signes, groupes horizontaux, groupes verticaux, superpositions, insertions, lacunes et zones incertaines. Cette approche évite de manipuler le MdC comme une chaîne opaque.

Ambiguïtés

Plusieurs encodages peuvent coexister. Un signe non identifié utilise un marqueur explicite accompagné d'une référence à l'objet, plutôt qu'un code inventé. Les restitutions, signes endommagés et signes omis sont annotés au niveau structurel.

Validation

Le validateur visualise simultanément :

- la chaîne MdC ;
- un rendu hiéroglyphique ;
- les boîtes correspondantes sur l'image ;
- l'ordre de lecture ;
- les alertes de syntaxe MdC.

Interopérabilité

Le module distingue le modèle interne de ses sérialisations. Il peut exporter MdC historique, Unicode format controls ou une représentation XML/TEI, dans la mesure où les constructions sont représentables.

Contrat, invariants et validation

L'entrée est une branche de `ReadingOrder` et un ensemble de sélections glyphiques. La sortie contient un arbre syntaxiquement valide, une sérialisation et une table d'alignement. Chaque feuille de type `sign`, `unknown`, `lacuna` ou `restoration` référence au moins une source ou explique l'absence de source observée. Un même glyphe ne peut apparaître deux fois dans une branche sans relation éditoriale explicite.

Le validateur de format contrôle parenthèses, opérateurs, codes et portée. Des tests aller-retour garantissent que parser puis sérialiser conserve l'arbre à équivalence définie. Les cas de test couvrent groupes imbriqués, signes superposés, insertions, cartouches, lacunes et ordre incertain. La validation scientifique compare le rendu à l'image, non seulement la chaîne attendue. Lorsqu'un export perd une nuance du modèle interne, il émet un avertissement structuré et ne prétend pas être sans perte.

État

Ce module appartient à l'architecture cible.

5.8. LoraGrammar

Mission

LoraGrammar produit segmentation, translittération, analyse morphologique et relations syntaxiques. Il ne choisit pas silencieusement une seule lecture lexicale.

Niveaux d'analyse

Le module distingue :

- valeurs possibles des signes ;
- translittération normalisée ;
- frontières de mots ;
- lemmes ;
- traits morphologiques ;
- fonctions syntaxiques ;
- structures de proposition ;
- phénomènes éditoriaux ou diachroniques.

Chaque niveau référence les segments graphiques correspondants.

Méthodes

Une première version peut être fondée sur règles, lexiques et patrons. Des modèles statistiques peuvent classer les analyses, mais doivent exposer leurs candidats. Le module peut utiliser époque, genre textuel et provenance comme contexte, à condition que ces métadonnées soient elles-mêmes qualifiées.

Gestion des variantes

Une segmentation concurrente engendre une branche d'analyse, non une écrasement. Les dépendances permettent de comparer les conséquences grammaticales de deux identifications glyphiques.

Sorties

Les sorties comprennent tokens, lemmes, traits, dépendances, constituants, diagnostics et scores. Un diagnostic précise s'il s'agit d'une impossibilité, d'une rareté ou d'une absence de règle.

Contrat, invariants et validation

Chaque analyse reçoit un encodage MdC, un profil de langue et une version de ressources. Elle retourne un graphe de tokens alignés et éventuellement plusieurs branches. Aucun token ne peut couvrir un segment absent sans être marqué comme restitution ou élément implicite. Les normalisations conservent la forme diplomatique, et les traits utilisent des vocabulaires versionnés.

L'évaluation distingue reconnaissance de valeurs, segmentation, lemmatisation, morphologie et syntaxe. Les jeux d'or doivent documenter les désaccords d'éditeurs au lieu d'imposer artificiellement une vérité unique. Une règle doit être testée sur des exemples positifs et négatifs, avec période et genre lorsque sa portée est limitée. Les diagnostics sont considérés comme une sortie scientifique : un échec explicable vaut mieux qu'une analyse complète mais incohérente. Le module doit aussi fonctionner en mode partiel sur une phrase lacunaire.

État

Ce module n'est pas encore implémenté.

5.9. LoraLexicon

Mission

LoraLexicon fournit des connaissances de référence : listes de signes, valeurs phonétiques, déterminatifs, lexèmes, graphies, noms propres et références bibliographiques.

Sources

Les ressources peuvent inclure la liste de Gardiner, des extensions, des dictionnaires et des index de noms propres. Chaque entrée indique sa source, son édition, ses droits et ses éventuelles divergences. Les références à Faulkner, Hannig ou d'autres ouvrages ne doivent jamais laisser croire que leur contenu est librement redistribuable.

Modèle

Une entrée lexicale peut contenir :

- identifiant pérenne ;
- lemme et translittération ;
- classe grammaticale ;
- sens ordonnés et attestations ;
- périodes et genres ;
- graphies ;
- déterminatifs ;
- renvois bibliographiques ;
- statut éditorial.

Une entrée de signe distingue catégorie Gardiner, description visuelle, valeurs, usages et variantes paléographiques.

API

Le module permet recherche exacte, normalisée, par préfixe, par valeur phonétique, par catégorie, par époque et par similarité de graphie. Les résultats sont paginés et citables.

Contrat, invariants et validation

Une réponse lexicale distingue la donnée de référence, les variantes éditoriales et les enrichissements propres au projet. Chaque sens ou valeur peut avoir sa propre source et sa propre portée chronologique. Les mises à jour créent une nouvelle édition de ressource ; elles ne changent pas rétroactivement les analyses qui ont utilisé une édition antérieure.

Les tests portent sur normalisation, homographes, renvois circulaires, codes inexistants et recherche multilingue. Des contrôles d'intégrité vérifient l'unicité des identifiants, la validité des citations et l'absence de liens orphelins. La qualité ne se réduit pas au nombre d'entrées : couverture, précision des sources, granularité et droits

d'usage sont publiés. Lorsqu'un dictionnaire sous droits ne peut être distribué, l'architecture conserve un connecteur ou une référence locale sans copier son contenu dans les exports publics.

État

Ce module relève de la cible. La première livraison doit privilégier des référentiels juridiquement réutilisables et un mécanisme d'import documenté.

5.10. LoraCorpus

Mission

LoraCorpus organise les textes de comparaison et recherche des parallèles. Il ne valide pas une lecture par simple similarité.

Unités de corpus

Un corpus contient des œuvres, témoins, passages, éditions et alignements. Il distingue le texte abstrait d'un témoin matériel. Chaque passage conserve bibliographie, provenance, datation, genre, langue, état éditorial et droits.

Index

Plusieurs index sont nécessaires :

- séquences Gardiner ;
- séquences MdC normalisées ;
- translittérations ;
- lemmes ;
- patrons morphosyntaxiques ;
- traductions ;
- métadonnées ;
- vecteurs visuels ou textuels, utilisés comme outils de rappel.

Recherche

Une requête peut inclure des jokers, lacunes, alternatives et contraintes de distance. Le score final expose ses composantes : similarité graphique, lexicale, syntaxique, chronologique et documentaire.

Risque de circularité

Un parallèle issu d'une édition moderne ne doit pas être présenté comme observation indépendante si cette édition dépend déjà de l'inscription étudiée. Les relations bibliographiques et généalogiques des éditions sont donc essentielles.

Contrat, invariants et validation

La requête structurée contient représentation recherchée, contraintes, filtres, nombre de résultats et version d'index. Chaque résultat référence un passage stable et détaille

le calcul du score. Les extraits respectent les licences ; un résultat peut signaler l'existence d'un passage sans en reproduire le texte intégral.

L'évaluation emploie des ensembles de requêtes et jugements de pertinence établis par des spécialistes. Elle publie rappel, précision aux premiers rangs et diversité des témoins. Des tests spécifiques vérifient lacunes, jokers, normalisations et exclusion d'une inscription de sa propre recherche. Les résultats doivent être reproductibles pour une version d'index. Lorsqu'un index évolue, les anciens rapprochements restent consultables avec leur version et peuvent être marqués stale, jamais réécrits silencieusement.

État

Ce module reste à construire. Il nécessite une politique de corpus, de licences et de normalisation avant tout index à grande échelle.

5.11. LoraReasoner

Mission

LoraReasoner confronte vision, validations, MdC, grammaire, lexique et corpus pour produire des conclusions argumentées. Il ne remplace pas les modules spécialisés et ne transforme pas une corrélation en preuve.

Modèle de raisonnement

Le raisonneur manipule des assertions reliées par :

- supports ;
- contradicts ;
- depends_on ;
- alternative_to ;
- derived_from ;
- requires_assumption.

Une conclusion expose ses prémisses, les règles appliquées, les contre-arguments, les alternatives et un niveau de confiance qualifié.

Types de règles

Les règles peuvent être :

- déterministes : incompatibilité géométrique ;
- heuristiques : forme fréquente dans une période ;
- linguistiques : accord ou régime ;
- documentaires : attestation dans un parallèle ;
- éditoriales : préférence d'une édition ;
- statistiques : classement par modèle.

Le type et la portée d'une règle sont toujours visibles.

Calcul par branches

Le moteur peut construire plusieurs scénarios cohérents. Il évite l'explosion combinatoire par seuils, contraintes et regroupement d'hypothèses équivalentes, sans supprimer les branches rejetées du dossier scientifique.

Sorties

Une sortie typique est : « lecture A préférée parce qu'elle concorde avec la forme, la syntaxe et deux parallèles ; lecture B conservée car la zone est endommagée ; lecture C rejetée par incompatibilité d'ordre ».

Contrat, invariants et validation

Le raisonneur reçoit un sous-graphe d'assertions et un jeu de règles versionné. Il retourne des scénarios, leurs conclusions et un graphe d'argumentation. Toute conclusion automatique possède au moins une chaîne de dépendances jusqu'à des observations ou autorités documentaires. Les règles statistiques ne sont pas converties en règles logiques, et une absence dans le corpus n'est pas traitée comme preuve d'impossibilité.

Les tests couvrent contradictions, cycles, prémisses manquantes, branches équivalentes et changement local d'hypothèse. Des « tests d'explication » vérifient que chaque phrase justificative correspond aux relations réellement calculées. L'évaluation par expert juge correction, exhaustivité des contre-arguments et utilité du classement. Le moteur doit pouvoir répondre à une interrogation contrefactuelle — « si G43 était retenu, quelles conclusions changeraient ? » — en réutilisant le graphe de dépendances.

État

Ce module appartient à la cible de long terme. Une première version fondée sur règles explicites et graphes de dépendance est préférable à un système opaque.

5.12. LoraTranslate

Mission

LoraTranslate produit une ou plusieurs traductions à partir d'une analyse sélectionnée. La traduction est une conclusion éditoriale, non une propriété directe des glyphes.

Sorties

Le module peut produire :

- traduction très littérale ;
- traduction philologique annotée ;
- traduction fluide ;
- glosses interlinéaires ;
- alternatives locales ;
- notes justificatives.

Chaque segment de traduction s'aligne sur les tokens et assertions sources.

Usage de modèles génératifs

Un modèle génératif peut proposer une formulation, mais il reçoit une entrée structurée et ne doit pas inventer de contenu absent. Sa sortie reste une hypothèse avec modèle, version, paramètres et prompt de système enregistrés dans la mesure permise. Les noms propres, lacunes et restitutions font l'objet de contraintes.

Validation

Le validateur compare traduction, translittération, analyse, image et parallèles. Une modification manuelle crée une nouvelle version et peut être accompagnée d'une note.

Qualité

L'évaluation combine fidélité philologique, explicitation des lacunes, cohérence terminologique, lisibilité et accord d'experts. Les scores automatiques génériques ne suffisent pas.

Contrat, invariants et validation

Une requête précise analyse source, langue cible, profil de traduction, public et contraintes terminologiques. Chaque unité traduite référence ses tokens sources. Une unité sans alignement est interdite, sauf ajout éditorial marqué. La sortie conserve les variantes et ne remplace pas la traduction humaine retenue.

Les tests automatiques ciblent noms propres, nombres, négations, lacunes, restitutions et cohérence de glossaire. Une évaluation en aveugle par spécialistes compare fidélité, formulation et transparence des incertitudes. Pour un moteur génératif, des tests adversariaux recherchent hallucinations de titres, filiations ou compléments plausibles. La publication exige une revue humaine et un manifeste du moteur utilisé. Un changement stylistique ne doit pas altérer les marqueurs scientifiques ; les crochets, points d'incertitude et notes sont générés depuis la structure, non confiés à la mémoire du modèle.

État

Ce module reste à implémenter après stabilisation des couches graphiques et linguistiques.

5.13. LoraSpeech

Mission

LoraSpeech fournit une restitution audio pédagogique ou expérimentale d'une translittération, d'une lecture conventionnelle ou d'une traduction.

Précautions

La vocalisation exacte de l'égyptien ancien n'est généralement pas connue. Toute sortie doit donc déclarer son régime :

- lecture égyptologique conventionnelle ;
- reconstruction proposée pour une phase de langue ;
- prononciation pédagogique ;
- lecture de la traduction moderne.

Le système évite de présenter une synthèse comme reconstitution certaine.

Entrées et sorties

L'entrée référence une translittération ou une traduction validée, une politique de prononciation, une langue, une voix et une vitesse. La sortie est un fichier audio avec alignement temporel et provenance.

Contrat, invariants et validation

Le profil de prononciation est un objet versionné qui décrit les règles de conversion et leurs sources. Le fichier audio référence exactement le texte vocalisé ; toute modification du texte impose une nouvelle synthèse. Les métadonnées embarquées et le manifeste signalent clairement le caractère conventionnel ou reconstruit de la lecture.

Les tests contrôlent la présence de tous les segments, l'alignement temporel, la prononciation des codes particuliers et l'accessibilité du lecteur. Une écoute experte évalue intelligibilité et conformité au profil, non l'« authenticité » absolue. Les sorties peuvent être désactivées pour des textes ou noms sensibles selon la politique du corpus.

État

Ce module est optionnel et postérieur aux fonctions scientifiques centrales.

5.14. LoraWorkflow

Mission

LoraWorkflow orchestre les modules, gère dépendances, reprise, cache, invalidation, erreurs et observabilité. Il ne contient pas de logique égyptologique propre.

Exécution

Un workflow est un graphe acyclique ou un processus à états. Chaque tâche possède entrées, sorties, politique de nouvelle tentative, délai, ressources et condition de déclenchement.

Les activités sont idempotentes : relancer une étape avec les mêmes entrées et paramètres ne doit pas créer des conclusions divergentes sans explication. L'adressage par empreinte permet de réutiliser les résultats.

Reprise

Une interruption conserve les étapes terminées. Une erreur locale sur un objet graphique ne doit pas nécessairement interrompre le document entier : l'objet reçoit error, tandis que les autres peuvent progresser.

Invalidation

Une nouvelle validation de signe invalide uniquement les MdC, analyses et traductions qui en dépendent. Les images et mises en page indépendantes restent valides.

Observabilité

Le module publie journaux structurés, durées, volumes, versions et erreurs. Les journaux techniques ne remplacent pas la provenance scientifique, mais y renvoient.

Contrat, invariants et validation

Le plan d'exécution est lui-même versionné. Une tâche ne démarre que si ses préconditions sont satisfaites et ses entrées résolues. Les clés d'idempotence empêchent les doublons accidentels ; les verrous sont assez fins pour autoriser des revues parallèles. L'annulation produit un état terminal documenté et ne supprime pas les sorties déjà achevées.

Les tests d'intégration simulent pannes, délais, reprise, exécution concurrente, événement dupliqué et migration de module. Un scénario de référence rejoue une inscription entière et compare le manifeste attendu. Les objectifs de service portent sur disponibilité, délai de démarrage et taux de reprise, tandis que les objectifs scientifiques portent sur l'exactitude des dépendances et de l'invalidation. Un tableau de bord doit distinguer échec technique, absence légitime de résultat et blocage en attente d'expertise.

État

Une première orchestration Phase 1 complète est implémentée : chargement, redressement, contraste, analyse de mise en page, overlay, extraction des vignettes, projection JSON historique, import des hypothèses et écriture d'un LoraDocument 1.0 valide. Elle fonctionne localement sous Windows 11 avec Python 3.12, NumPy et OpenCV.

La branche expérimentale impose désormais une arborescence de sortie `output\<version>\<test>\`. Une nouvelle version ou un nouveau test ne doit jamais écraser les artefacts précédents. Le rapport d'exécution enregistre nom du fichier, version du programme, début, fin, durée, paramètres, nombres d'objets par classe ou couleur, candidats Gardiner et scores. Les gros corpus de référence ne sont pas copiés dans chaque dossier de résultat ; ils sont référencés par manifeste, version et empreinte.

6. Cycle de vie d'une inscription

6.1. Création du dossier

Un opérateur importe les photographies. LoraCapture crée le document, calcule les empreintes, enregistre les droits et relie les vues. L'original est figé.

6.2. Préparation visuelle

LoraImage produit une image normalisée. L'opérateur vérifie perspective, orientation et lisibilité. Les matrices permettent toujours de revenir aux coordonnées originales.

6.3. Segmentation

LoraLayout propose registres, lignes, colonnes, cadrats, cartouches et zones endommagées. Une revue corrige les erreurs structurantes avant la reconnaissance.

6.4. Ordre de lecture

LoraDirection produit une ou plusieurs séquences. Les ambiguïtés sont maintenues sous forme de graphes alternatifs.

6.5. Reconnaissance glyphique

LoraGlyph extrait chaque région, la classe et propose des candidats Gardiner. Les objets incertains entrent dans une file de revue. Les validations créent des événements.

6.6. Encodage

LoraMdC assemble les signes sélectionnés. Le rendu et l'image sont vérifiés côte à côte. Les portions non lues restent explicitement marquées.

6.7. Analyse linguistique

LoraGrammar et LoraLexicon proposent translittérations, segmentations, lemmes et constructions. Les branches incompatibles sont comparées.

6.8. Recherche de parallèles

LoraCorpus cherche des séquences ou formulations proches. Les résultats sont des éléments de preuve possibles, jamais une validation automatique.

6.9. Raisonnement et traduction

LoraReasoner synthétise les arguments. LoraTranslate génère des traductions alignées et annotées. Un expert sélectionne ou reformule la version publiée.

6.10. Publication et révision

Une révision est figée, accompagnée de ses sources, licences, versions et signatures. Toute correction ultérieure crée une nouvelle révision. L'ancienne demeure citée et consultable selon les droits.

Les statuts suggérés sont :

[illegible]

7. Gestion des hypothèses et de l'incertitude

7.1. Vocabulaire

Loracrafft distingue :

- **score moteur** : valeur interne de classement ;
- **probabilité calibrée** : estimation validée sur un jeu comparable ;
- **confiance scientifique** : appréciation argumentée ;
- **certitude éditoriale** : choix d'une édition pour publication ;
- **désaccord** : coexistence de décisions incompatibles.

Ces notions ne sont jamais fusionnées dans un champ confidence non qualifié.

7.2. Hypothèses

Une hypothèse contient une proposition, sa cible, son auteur, ses preuves, ses dépendances et son statut. Les statuts minimaux sont :

- proposed ;
- accepted ;
- rejected ;
- superseded ;
- deferred ;
- undecidable.

L'acceptation est toujours relative à une politique ou une édition. Une hypothèse peut être acceptée dans une édition de travail et rester contestée scientifiquement.

7.3. Propagation

Une incertitude en amont doit rester visible en aval. Si un signe possède deux lectures, deux analyses grammaticales et deux traductions peuvent être produites. Une synthèse peut agréger les branches, mais elle doit indiquer ce qui varie.

7.4. Seuils

Les seuils servent au routage :

- score élevé : proposition directe à confirmer ;
- score moyen : revue prioritaire ;
- score faible : candidat affiché sans sélection ;
- aucun candidat : needs_review.

Un seuil ne transforme pas une sortie automatique en fait.

7.5. Lacunes et restitutions

Une lacune observée est un objet géométrique. Une restitution est une hypothèse textuelle distincte. Le texte restauré doit être marqué dans tous les exports et relié aux arguments qui le soutiennent.

7.6. Désaccord humain

Le système conserve chaque vote et justification. Les indicateurs de consensus sont calculés à part. Une décision arbitrale référence les avis antérieurs et explique le choix.

7.7. Calibration

Les moteurs qui publient une probabilité doivent être calibrés sur des données proches du domaine d'usage. Les rapports sont ventilés par support, époque, état de conservation et qualité d'image. Une alerte est produite hors domaine connu.

8. Traçabilité et provenance

8.1. Modèle

La provenance suit un modèle proche de W3C PROV :

- **entités** : images, objets, hypothèses, encodages, traductions ;
- **activités** : import, transformation, détection, validation, export ;
- **agents** : personnes, logiciels, modèles, organisations.

Les relations essentielles sont `wasGeneratedBy`, `used`, `wasAssociatedWith`, `wasDerivedFrom` et `wasAttributedTo`.

8.2. Activité automatique

```
{
  "id": "urn:lora:activity:a-55",
  "type": "GlyphRecognition",
  "module": {"name": "loraglyph", "version": "1.0.1"},
  "engine": {
    "name": "template",
    "catalog_sha256": "..."
  },
  "parameters": {"top_k": 3, "margin": 4},
  "input_refs": [
    "urn:lora:image:i-2",
    "urn:lora:layout-object:lo-42"
  ],
  "started_at": "2026-07-29T10:03:12Z",
  "ended_at": "2026-07-29T10:03:13Z",
  "status": "succeeded",
  "output_refs": ["urn:lora:glyph-hypothesis:gh-42"]
}
```

8.3. Journal append-only

Les validations, publications et changements de statut sont enregistrés dans un journal append-only. Une projection courante peut afficher la dernière décision, mais doit pouvoir être reconstruite depuis les événements.

Le journal de corrections JSONL de LoraGlyph a constitué le premier prototype de ce principe. LoraDocument 1.0 représente désormais les validations comme objets autonomes et conserve en parallèle un journal append-only d'événements.

8.4. Empreintes

Une empreinte doit couvrir les octets d'un artefact. Pour un objet JSON, une canonicalisation définie précède le calcul. Les empreintes permettent de prouver l'identité des entrées et de détecter les altérations ; elles ne remplacent pas la signature ou l'archivage.

8.5. Citabilité

Une publication fournit :

- identifiant du document ;
- identifiant de révision ;
- date ;
- auteurs et contributeurs ;
- licence ;
- empreinte du manifeste ;
- format de citation ;
- liens vers sources et dépendances autorisées.

8.6. Données sensibles

Les journaux ne doivent pas incorporer de secrets ni contourner les restrictions des images. Les aperçus, vignettes et exports sont eux-mêmes des artefacts soumis aux droits.

9. Architecture des corpus

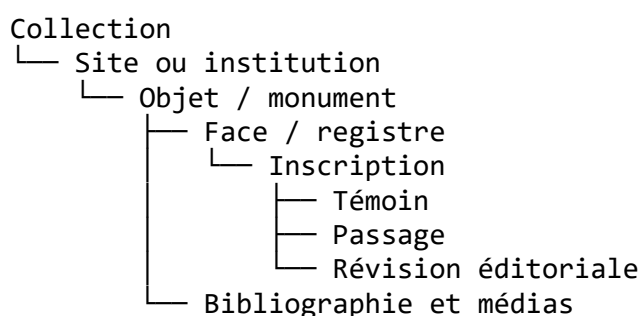
9.1. Séparation des corpus

Quatre familles sont distinguées :

1. **corpus documentaire** : images et inscriptions étudiées ;
2. **corpus de référence** : éditions, lexiques et parallèles ;
3. **corpus d'annotation** : validations humaines qualifiées ;
4. **corpus d'apprentissage et d'évaluation** : jeux dérivés, figés et versionnés.

Un même objet peut appartenir à plusieurs familles, mais ce statut est explicite.

9.2. Hiérarchie documentaire



La hiérarchie reste flexible grâce aux relations. Un fragment peut être rattaché hypothétiquement à plusieurs monuments.

9.3. Normalisation

Les corpus conservent la forme diplomatique et fournissent des vues normalisées. La normalisation indique ses règles : translittération, ponctuation, variantes de MdC, Unicode, casse, suffixes Gardiner et traitement des lacunes.

9.4. Versionnement

Chaque livraison de corpus possède un manifeste, une version, une licence, des statistiques et une empreinte. Un dataset d'apprentissage liste exactement les exemples inclus, leurs sources et leurs partitions.

9.5. Prévention des fuites

Les variantes d'un même objet ou d'une même photographie doivent rester dans la même partition. Les parallèles quasi identiques nécessitent un groupe documentaire afin d'éviter qu'un test ne mesure la mémorisation.

9.6. Gouvernance

Toute contribution suit une politique :

- critères d'admission ;

- niveau minimal de validation ;
- licences compatibles ;
- gestion des demandes de retrait ;
- attribution ;
- contrôle des biais géographiques, chronologiques et matériels.

9.7. Ressources disponibles au 17 août 2026

Trois ressources Raymond Monfort ont été examinées pour préparer le corpus de référence :

- le site et son archive locale `RaymondMonfort.zip` ;
- la page d'index `IndexA-a.html` ;
- le classeur catalogue `glyphes_MdC.xlsx`.

L'extraction de l'archive a révélé 6 722 fichiers glyphiques. Ce total doit être ventilé avant toute publication : nombre de codes Gardiner distincts, variantes graphiques par code, doublons binaires, formats, dimensions, transparence, fichiers sans code interprétable et relations avec les entrées MdC. Une quantité de fichiers n'est pas une couverture scientifique.

Le site ayant été réalisé par Didier Morandi pour Raymond Monfort, la provenance technique est connue ; les droits de redistribution scientifique doivent néanmoins être explicités dans un manifeste. La source doit être attribuée et sa licence distinguée de celle du code Loracraft.

Un lot de 160 scans est également disponible pour les essais sur documents réels. Ces scans ne doivent pas être répartis aléatoirement image par image entre entraînement et test. Toutes les vues, recadrages et dérivés d'une même inscription restent dans la même partition documentaire afin d'éviter une fuite de données.

Le corpus doit donc distinguer :

1. les gabarits propres ou rendus numériques ;
2. les variantes graphiques provenant du site Monfort ;
3. les crops issus des scans réels ;
4. les exemples validés manuellement ;
5. les cas réservés définitivement à l'évaluation.

10. LoraSDK

10.1. Finalité

LoraSDK est le kit de développement de Loracrafft. Il fournit aux contributeurs les types, interfaces, validateurs, outils de test et exemples nécessaires pour étendre la plateforme sans rompre ses principes scientifiques.

Le SDK ne constitue pas un nouveau moteur métier. Il est la frontière stable entre le noyau et les extensions. Il doit permettre de développer indépendamment :

- un moteur pour un module existant ;
- un nouveau module ;
- un connecteur de corpus ou de lexique ;
- un ensemble de règles pour LoraReasoner ;
- un importateur ou un exportateur ;
- une interface de revue ;
- un outil de migration.

Le SDK de référence est d'abord proposé en Python, afin de rester proche des prototypes actuels. Les contrats étant définis par JSON Schema et OpenAPI, des implémentations dans d'autres langages peuvent être ajoutées.

10.2. Contenu du SDK

Une distribution de LoraSDK contient au minimum :

lorasdk/	
├ models/	types LoraDocument et objets scientifiques
├ protocols/	interfaces de modules, moteurs et corpus
├ validation/	JSON Schema et validateurs métier
├ provenance/	constructeurs d'activités et d'agents
├ testing/	fixtures, faux moteurs et tests de conformité
├ serialization/	lecture, écriture et canonicalisation
├ errors/	codes d'erreur communs
└ cli/	validation et diagnostic d'une extension

Le SDK publie une matrice de compatibilité entre sa version, les versions de schémas et le noyau Loracrafft. Une extension déclare ses contraintes de compatibilité dans un manifeste ; elle ne se fie pas à des imports internes non documentés.

10.3. Manifeste d'extension

Toute extension possède un manifeste lisible sans exécuter son code :

```
{  
  "$schema": "https://loracrafft.org/schemas/extension-manifest/1.0",  
  "extension_id": "org.example.glyph-transformer",  
  "name": "ExampleGlyphTransformer",  
  "extension_type": "engine",  
}
```

```

"implements": "LoraGlyph.GlyphRecognizer",
"version": "1.0.0",
"sdk": ">=1.0,<2.0",
"entry_point": "example_glyph:Recognizer",
"capabilities": {
    "input_media": ["image/png", "image/jpeg"],
    "sign_references": ["gardiner"],
    "supports_batch": true
},
"resources": {
    "model_id": "example-glyph-v1",
    "model_sha256": "..."
},
"license": "Apache-2.0"
}

```

Le manifeste déclare aussi, lorsque nécessaire, accès réseau, ressources minimales, modèles, corpus requis, langues, licences et traitement éventuel de données sensibles. Le noyau peut ainsi refuser une extension incompatible avant son exécution.

10.4. Créer un nouveau moteur

Un moteur implémente une capacité spécialisée à l'intérieur d'un module. Un nouveau reconnaiseur de glyphes n'a donc pas à réimplémenter LoraGlyph, les validations humaines ou l'écriture du LoraDocument.

Exemple conceptuel :

```

from lorasdk.glyph import Candidate, GlyphRecognizer, GlyphRequest

class MyGlyphRecognizer(GlyphRecognizer):
    engine_id = "org.example.my-glyph-recognizer"

    def capabilities(self) -> dict:
        return {
            "sign_references": ["gardiner"],
            "supports_batch": False,
            "score_kind": "engine_confidence"
        }

    def recognize(
        self,
        request: GlyphRequest,
        *,
        top_k: int = 3
    ) -> list[Candidate]:
        # Le moteur peut retourner une liste vide.
        ...

```

La procédure recommandée est :

1. choisir l'interface la plus étroite ;
2. créer le manifeste ;

3. implémenter *capabilities*, la méthode métier et le diagnostic de santé ;
4. utiliser les types du SDK plutôt que des dictionnaires non validés ;
5. joindre la provenance du modèle ou du catalogue ;
6. exécuter la suite de conformité ;
7. publier les limites et métriques du moteur ;
8. fournir un exemple reproductible.

Un moteur ne doit jamais écrire directement dans les collections d'un autre module. Il retourne un résultat typé ; le module construit les objets scientifiques et l'activité.

10.5. Créer un nouveau corpus

Un corpus peut être distribué comme paquet versionné ou exposé par un connecteur. Dans les deux cas, le contributeur fournit :

- un identifiant stable de corpus ;
- un manifeste de version ;
- la licence et les règles de citation ;
- le modèle œuvre-témoin-passage ;
- les profils de normalisation ;
- un mécanisme de résolution des identifiants ;
- des statistiques de couverture ;
- les relations entre éditions ;
- un index reproductible ou les moyens de le reconstruire.

Interface conceptuelle :

```
class CorpusProvider:
    def manifest(self) -> CorpusManifest: ...
    def get_passage(self, passage_id: str) -> Passage: ...
    def search(self, query: CorpusQuery) -> SearchPage: ...
    def resolve(self, external_id: str) -> list[CorpusReference]: ...
```

Le SDK fournit des tests contre les identifiants orphelins, les citations absentes, les réponses non déterministes et les extraits incompatibles avec la licence. Un corpus ne doit pas masquer ses transformations : forme diplomatique, forme normalisée et index sont distingués.

10.6. Créer un nouveau module

Un nouveau module est justifié lorsqu'une responsabilité scientifique nouvelle ne peut pas être exprimée comme moteur d'un module existant. Il doit lire des collections déclarées et ajouter ses propres objets sans modifier les couches antérieures.

```
class ExampleModule(LoraModule):
    module_id = "LoraExample"

    def capabilities(self) -> ModuleCapabilities: ...
```

```

def validate_input(
    self,
    document: LoraDocument,
    selection: Selection
) -> list[Diagnostic]:
    ...

def run(
    self,
    context: RunContext,
    document: LoraDocument,
    selection: Selection
) -> ModuleResult:
    ...

```

La proposition d'un module doit documenter :

- sa question scientifique ;
- les collections lues et produites ;
- ses invariants ;
- ses erreurs et états sans résultat ;
- ses dépendances et événements d'invalidation ;
- son schéma JSON ;
- son protocole d'évaluation ;
- sa stratégie de migration ;
- au moins un scénario de bout en bout.

Le nom Lora... est réservé aux modules de premier niveau acceptés dans l'architecture officielle. Une extension expérimentale utilise d'abord un identifiant qualifié par son organisation.

10.7. Créer un nouveau Reasoner

Un Reasoner peut être un ensemble de règles, un moteur de contraintes, un raisonneur probabiliste ou un orchestrateur de scénarios. Il doit toutefois produire le même graphe d'argumentation.

Une règle déclare :

- son identifiant et sa version ;
- son type : déterministe, heuristique, linguistique, documentaire ou statistique ;
- les types de prémisses attendus ;
- la conclusion produite ;
- son domaine de validité ;
- sa source scientifique ;
- les conditions qui l'invalident ;
- son niveau de priorité éventuel.

```

class ReasoningRule:
    rule_id = "org.example.rule.orientation-agreement"
    def applies(self, graph: AssertionGraph) -> bool: ...
    def evaluate(self, graph: AssertionGraph) -> RuleResult: ...

```

RuleResult ne contient pas seulement un score. Il fournit les assertions utilisées, la relation de soutien ou de contradiction, les avertissements et une explication structurée. Les règles sont testées séparément sur cas positifs, négatifs et limites. Un raisonneur doit également réussir les tests de contrefactuel et ne jamais transformer l'absence de preuve en preuve d'absence sans règle explicite.

10.8. Créer un importateur ou un exportateur

Un adaptateur convertit un format externe vers le modèle interne ou inversement. Il déclare les pertes possibles. L'import conserve le fichier original et crée une activité de migration. L'export ne prétend pas être réversible si le format cible ne représente pas toutes les nuances.

Les adaptateurs prioritaires concernent :

- JSON historiques de LoraLayout et LoraGlyph ;
- MdC ;
- Unicode hiéroglyphique ;
- TEI/EpiDoc ;
- IIIF ;
- JSON-LD/PROV-O ;
- formats d'annotation d'image.

10.9. Tests de conformité

Le SDK fournit une commande conceptuelle :

```
lora-sdk check-extension ./mon-extension
```

Elle vérifie manifeste, imports, schémas, erreurs, provenance, déterminisme annoncé, cas sans résultat et absence de mutation des entrées. Des fixtures minimales couvrent image valide, objet aux limites, document partiel, identifiant inconnu, version incompatible et interruption.

Une extension est déclarée conforme lorsqu'elle passe les tests techniques. Cette conformité ne vaut pas validation scientifique : celle-ci nécessite un corpus d'évaluation, des métriques et une revue propres à son domaine.

10.10. Documentation et contribution

Toute contribution comprend :

- guide d'installation ;
- exemple minimal ;
- référence des paramètres ;

- limites connues ;
- politique de versionnement ;
- journal des changements ;
- tests ;
- licence du code et des données ;
- personne ou équipe responsable.

Les exemples doivent être exécutables hors ligne lorsque les dépendances le permettent. Une contribution qui appelle un service distant fournit un faux service pour les tests et décrit les données transmises.

11. Interfaces publiques : API internes

11.1. Portée et définition

Les API internes sont **publiques au sens architectural** : elles sont documentées, versionnées, testables et utilisables par tout module autorisé. Le mot « interne » signifie qu'elles appartiennent au système Loracraft ; il ne signifie ni « provisoire », ni « réservé à un fichier Python particulier ».

La même sémantique doit pouvoir être exposée par plusieurs transports :

- appel direct d'une fonction ou d'un protocole Python ;
- commande exécutée par la CLI ;
- lecture et écriture de fichiers JSON ou JSONL ;
- API HTTP sur le poste local ;
- service distant dans une installation connectée ;
- file de messages ou flux d'événements pour les traitements distribués.

Un module ne doit pas dépendre inutilement du transport. LoraGlyph, par exemple, reçoit le même contrat scientifique lorsqu'il est appelé depuis la CLI, une interface graphique ou un service HTTP. Les URL décrites dans ce chapitre sont donc conceptuelles : elles fixent une responsabilité et une structure d'échange, non l'obligation de déployer un serveur Internet.

11.2. Principes généraux

Les opérations sont :

- **non destructives** : elles ajoutent des objets ou créent une révision ;
- **idempotentes lorsque possible** : une même demande ne crée pas de doublon accidentel ;
- **asynchrones pour les traitements longs** ;
- **paginées pour les collections volumineuses** ;
- **munies d'identifiants de requête et de corrélation** ;
- **explicites sur les versions de contrat, de module, de moteur et de corpus** ;
- **capables de retourner des résultats partiels** ;
- **compatibles avec un fonctionnement hors ligne** lorsque la capacité ne dépend pas d'une ressource extérieure ;
- **déterministes lorsque le moteur le revendique**, avec paramètres et empreintes identiques ;
- **révocables par une nouvelle décision**, jamais par effacement silencieux d'un fait scientifique.

Une API ne renvoie pas seulement une valeur finale. Elle expose également l'activité qui l'a produite, les entrées utilisées, les avertissements, les objets sans résultat et les informations nécessaires à la reproduction.

11.3. Formes d'interface

Interface Python

L'interface Python convient aux modules exécutés dans le même processus. Elle emploie des objets typés ou des dictionnaires validés contre les mêmes schémas JSON que les autres transports.

```
from typing import Protocol

class LoraModule(Protocol):
    def manifest(self) -> dict: ...
    def capabilities(self) -> list[dict]: ...
    def validate_request(self, request: dict) -> list[dict]: ...
    def run(self, context: dict, request: dict) -> dict: ...
    def cancel(self, job_id: str) -> dict: ...
    def health(self) -> dict: ...
```

run retourne un résultat contenant objets à ajouter, activité et diagnostics. Il ne reçoit jamais l'autorisation implicite de modifier le LoraDocument fourni par l'appelant.

Interface CLI

La CLI traduit les arguments en requêtes structurées et sérialise les mêmes réponses. Elle ne contient pas une seconde implémentation métier. Les sorties machine utilisent JSON ; les messages destinés à l'utilisateur peuvent être localisés.

```
loracrafft run image.png -o output
loracrafft check document.json
loracrafft validate document.json <hypothesis_id> --action amend --gardiner
G17
```

Interface HTTP

L'interface HTTP utilise JSON UTF-8 pour les métadonnées. Les images et artefacts volumineux sont transmis ou référencés séparément ; ils ne sont pas incorporés systématiquement en Base64 dans le document.

Les installations locales peuvent écouter uniquement sur localhost. Une installation distante ajoute authentification, contrôle d'accès, chiffrement, quotas et journal d'audit sans modifier le contrat scientifique.

11.4. Enveloppe commune des requêtes

Toute requête importante possède une enveloppe commune :

```
{
  "$schema": "https://loracrafft.org/schemas/request/1.0",
  "request_id": "urn:lora:request:r-44",
  "correlation_id": "urn:lora:correlation:c-18",
  "submitted_at": "2026-08-17T10:03:00Z",
  "submitted_by": {
    "kind": "human",
```

```

    "id": "urn:lora:person:didier"
  },
  "document_id": "urn:lora:document:d-1",
  "revision_id": "urn:lora:revision:r-7",
  "contract_version": "1.0",
  "locale": "fr-FR",
  "idempotency_key": "sha256:...",
  "payload": {}
}

```

request_id identifie la demande. correlation_id relie les appels appartenant à une même opération utilisateur. revision_id fixe l'état du document lu par le traitement. idempotency_key protège contre les répétitions dues à un double clic, une reprise réseau ou un consommateur d'événements redémarré.

La même clé d'idempotence accompagnée du même contenu retourne le traitement déjà créé. La même clé accompagnée d'un contenu différent produit IDEMPOTENCY_CONFLICT.

11.5. Manifeste et capacités d'un module

Avant d'exécuter un module, l'orchestrateur peut interroger son manifeste :

```

{
  "module_id": "org.loracrafft.glyph",
  "module_name": "LoraGlyph",
  "module_version": "1.0.1",
  "contract_versions": ["1.0"],
  "capabilities": [
    {
      "capability_id": "recognize-gardiner",
      "input_types": ["LayoutObject"],
      "output_types": ["GlyphHypothesis"],
      "supports_partial_results": true,
      "supports_cancellation": true,
      "offline": true
    }
  ],
  "engines": [
    {
      "engine_id": "org.loracrafft.template-recognizer",
      "engine_version": "1.0.1",
      "model_id": null,
      "catalogue_required": true
    }
  ]
}

```

Le manifeste permet de refuser proprement une demande incompatible avant le lancement. Il distingue version du module, version du moteur, version du modèle et version du catalogue.

Endpoints conceptuels :

```
GET /internal/v1/modules
GET /internal/v1/modules/{module_id}
GET /internal/v1/modules/{module_id}/capabilities
GET /internal/v1/modules/{module_id}/health
```

health décrit l'état technique du service. Il ne constitue jamais une mesure de qualité scientifique.

11.6. Documents, artefacts et révisions

Les opérations documentaires minimales sont :

```
POST /internal/v1/documents
GET /internal/v1/documents/{document_id}
GET /internal/v1/documents/{document_id}/revisions
GET /internal/v1/documents/{document_id}/revisions/{revision_id}
POST /internal/v1/documents/{document_id}/revisions

POST /internal/v1/artifacts
GET /internal/v1/artifacts/{artifact_id}
GET /internal/v1/artifacts/{artifact_id}/metadata
```

La création d'une révision reçoit un parent explicite et un ensemble d'ajouts :

```
{
  "parent_revision_id": "urn:lora:revision:r-7",
  "expected_current_revision_id": "urn:lora:revision:r-7",
  "additions": {
    "glyph_hypotheses": [
      "urn:lora:glyph-hypothesis:gh-42"
    ],
    "activities": [
      "urn:lora:activity:a-91"
    ]
  },
  "reason": "Reconnaissance Gardiner sur les cadrats validés"
}
```

expected_current_revision_id fournit un verrou optimiste. Si un autre utilisateur a publié une nouvelle révision entre-temps, l'API retourne REVISION_CONFLICT. Elle ne remplace pas silencieusement la révision concurrente.

Un artefact binaire est identifié par une empreinte et décrit par ses métadonnées : type MIME, dimensions, taille, droits, origine et transformations. Un stockage peut dédupliquer deux contenus identiques, mais les deux événements de dépôt restent distincts.

11.7. Création et cycle de vie d'un traitement

Un traitement long est représenté par un Job. Exemple de demande :

```
{
  "$schema": "https://loracrafft.org/schemas/job-request/1.0",
  "request_id": "urn:lora:request:r-44",
}
```

```

"correlation_id": "urn:lora:correlation:c-18",
"document_id": "urn:lora:document:d-1",
"revision_id": "urn:lora:revision:r-7",
"module_id": "org.loracraft.glyph",
"capability_id": "recognize-gardiner",
"selection": {
  "layout_object_ids": [
    "urn:lora:layout-object:lo-42"
  ]
},
"engine": {
  "engine_id": "org.loracraft.template-recognizer",
  "engine_version": "1.0.1",
  "catalogue_id": "org.loracraft.monfort-glyphs",
  "catalogue_version": "1.0"
},
"parameters": {
  "top_k": 3,
  "reject_below": 0.0
},
"idempotency_key": "sha256:..."
}

```

Réponse immédiate :

```

{
  "job_id": "urn:lora:job:j-44",
  "status": "queued",
  "submitted_at": "2026-08-17T10:03:00Z",
  "status_url": "/internal/v1/jobs/urn:lora:job:j-44",
  "events_url": "/internal/v1/jobs/urn:lora:job:j-44/events"
}

```

Les états autorisés sont :

```

queued → running → succeeded
                ↘ partial
                ↘ failed
queued/running → cancelling → cancelled

```

Un traitement partial a produit au moins un résultat exploitable et au moins une erreur locale. failed signifie qu'aucune sortie contractuellement utilisable n'a été produite. Une absence légitime de candidat n'est ni partial ni failed.

Endpoints conceptuels :

```

POST /internal/v1/jobs
GET  /internal/v1/jobs/{job_id}
GET  /internal/v1/jobs/{job_id}/results
GET  /internal/v1/jobs/{job_id}/events
POST /internal/v1/jobs/{job_id}/cancel
POST /internal/v1/jobs/{job_id}/resume

```

L'annulation est coopérative. Elle empêche les nouvelles unités de travail de démarrer, mais conserve les objets déjà produits et leur activité. Une reprise référence le traitement précédent et expose les étapes réutilisées.

11.8. Progression et résultat d'un traitement

La progression est quantitative lorsque le volume total est connu :

```
{
  "job_id": "urn:lora:job:j-44",
  "status": "running",
  "phase": "recognition",
  "completed_units": 38,
  "total_units": 165,
  "progress_ratio": 0.230303,
  "started_at": "2026-08-17T10:03:02Z",
  "updated_at": "2026-08-17T10:03:18Z",
  "warnings_count": 2,
  "errors_count": 0
}
```

Le résultat final sépare les réussites, les absences de candidat et les erreurs :

```
{
  "job_id": "urn:lora:job:j-44",
  "status": "partial",
  "activity_id": "urn:lora:activity:a-91",
  "results": [
    {
      "input_id": "urn:lora:layout-object:lo-42",
      "status": "succeeded",
      "output_ids": ["urn:lora:glyph-hypothesis:gh-42"]
    },
    {
      "input_id": "urn:lora:layout-object:lo-43",
      "status": "no_candidate",
      "output_ids": ["urn:lora:glyph-hypothesis:gh-43"]
    },
    {
      "input_id": "urn:lora:layout-object:lo-44",
      "status": "failed",
      "error": {
        "code": "INVALID_GEOMETRY",
        "retryable": false
      }
    }
  ],
  "summary": {
    "requested": 3,
    "succeeded": 1,
    "no_candidate": 1,
    "failed": 1
  }
}
```

Même sans candidat, gh-43 existe et documente classification, moteur, catalogue et paramètres. L'absence de reconnaissance reste donc un résultat scientifique traçable.

11.9. Collections, filtres et pagination

Les collections importantes ne sont jamais retournées sans limite. La pagination utilise un curseur opaque lié à un instantané ou à une révision :

```
GET /internal/v1/documents/{document_id}/objects
    ?revision_id=urn:lora:revision:r-7
    &type=LayoutObject
    &object_type=QUADRAT
    &status=reviewed
    &page_size=100
    &cursor=opaque-token
```

Réponse :

```
{
  "items": [],
  "page_size": 100,
  "next_cursor": "opaque-next-token",
  "snapshot_revision_id": "urn:lora:revision:r-7",
  "total_count": 165
}
```

total_count peut être absent lorsque son calcul est coûteux. Un curseur n'est pas construit par le client et peut expirer. La pagination ne doit pas mélanger silencieusement deux révisions du document.

Les filtres acceptés sont énumérés par contrat. Un paramètre inconnu produit une erreur explicite plutôt que d'être ignoré.

11.10. Hypothèses et validations humaines

Les hypothèses et validations possèdent des endpoints distincts :

```
POST /internal/v1/hypotheses
GET /internal/v1/hypotheses/{hypothesis_id}
GET /internal/v1/hypotheses/{hypothesis_id}/validations
```

```
POST /internal/v1/validations
GET /internal/v1/validations/{validation_id}
```

Exemple de validation :

```
{
  "$schema": "https://loracraft.org/schemas/human-validation/1.0",
  "target_id": "urn:lora:glyph-hypothesis:gh-42",
  "action": "amend",
  "selected_value": "gardiner:G17",
  "certainty": "high",
  "agent": {
    "kind": "human",
```

```

    "id": "urn:lora:person:didier"
  },
  "rationale": "Forme compatible avec la chouette ; comparaison avec le si
gne voisin.",
  "evidence_refs": [
    "urn:lora:image:i-normalized",
    "urn:lora:layout-object:lo-42"
  ],
  "expected_revision_id": "urn:lora:revision:r-7"
}

```

L'API crée une nouvelle validation et une nouvelle révision. Elle ne possède pas d'opération PUT permettant de réécrire la validation antérieure. Une rectification est une nouvelle validation qui peut référencer `replaces` ou `supersedes_for_purpose`.

Deux validations concurrentes restent toutes deux visibles. Une opération distincte peut calculer accord ou consensus, sans transformer automatiquement la majorité en vérité.

11.11. Provenance et explication

Tout objet scientifique peut être interrogé :

```

GET /internal/v1/objects/{object_id}
GET /internal/v1/objects/{object_id}/provenance
GET /internal/v1/objects/{object_id}/dependencies
GET /internal/v1/objects/{object_id}/dependents
GET /internal/v1/objects/{object_id}/explanation

```

La provenance retourne entités, activités, agents et relations. `dependencies` sert notamment à déterminer quelles sorties doivent être recalculées après une nouvelle validation. `explanation` produit une vue lisible à partir de ces données ; elle ne remplace pas le graphe de provenance.

Une explication de candidat Gardiner peut contenir :

- image et région utilisées ;
- transformation géométrique appliquée ;
- classification primaire ;
- catalogue et variantes comparées ;
- scores de Hu, silhouette, contour ou modèle appris ;
- candidats écartés ;
- seuils et règle de fusion ;
- validations humaines disponibles ;
- parallèles de corpus éventuellement consultés.

11.12. API de corpus et version connectée au TLA

LoraCorpus expose une interface générique afin que le cœur ne dépende pas d'un fournisseur particulier :

```

GET /internal/v1/corpora
GET /internal/v1/corpora/{corpus_id}/manifest
POST /internal/v1/corpora/{corpus_id}/search
GET /internal/v1/corpus-matches/{match_id}

```

Une requête de corpus contient la représentation recherchée et son contexte :

```

{
  "$schema": "https://loracrafft.org/schemas/corpus-search-request/1.0",
  "corpus_id": "de.bbaw.tla",
  "corpus_release": "20",
  "query": {
    "kind": "lemma_or_sequence",
    "transliteration_candidates": ["mꜣc-hrw"],
    "gardiner_sequences": [
      ["G17", "D36", "D21"]
    ],
    "mdc_candidates": ["G17-D36-D21"]
  },
  "context": {
    "language_stage": "Middle Egyptian",
    "object_type": "stela",
    "dating": null
  },
  "page_size": 20,
  "include": ["lemmata", "attestations", "translations", "metadata"]
}

```

La réponse ne valide pas automatiquement la lecture :

```

{
  "corpus_id": "de.bbaw.tla",
  "corpus_release": "20",
  "retrieved_at": "2026-08-17T10:14:00Z",
  "matches": [
    {
      "match_id": "urn:lora:corpus-match:cm-8",
      "external_id": "tla:stable-id",
      "match_kind": "lemma_attestation",
      "score": 0.82,
      "score_kind": "loracrafft-context-ranking",
      "evidence": {
        "matched_transliteration": "mꜣc-hrw",
        "stable_url": "https://thesaurus-linguae-aegyptiae.de/..."
      },
      "status": "proposed"
    }
  ],
  "next_cursor": null
}

```

Le score appartient au connecteur Loracrafft, sauf déclaration contraire. Il ne doit pas être présenté comme une confiance publiée par le TLA.

Le connecteur TLA suit les règles suivantes :

- utiliser un service officiel, une publication de données autorisée ou une coopération explicite ;
- ne pas supposer qu'une page Web constitue une API stable ;
- ne pas aspirer un sous-corpus en contradiction avec les conditions d'utilisation ;
- enregistrer édition du corpus, version de l'interface ou de l'API, date d'accès et licence ;
- conserver les identifiants externes et citations ;
- mettre en cache uniquement ce qui est permis, avec empreinte et politique d'expiration ;
- marquer stale un rapprochement calculé avec une ancienne édition au lieu de le réécrire ;
- tolérer l'indisponibilité réseau sans bloquer les modules visuels locaux.

Le TLA fournit un corpus de textes lemmatisés et des listes de lemmes ; il ne reçoit pas une photographie pour reconnaître directement ses glyphes. La version connectée intervient après les hypothèses visuelles afin de rechercher lemmes, attestations, séquences et parallèles. Une présence dans le corpus renforce une hypothèse ; une absence ne prouve pas son impossibilité.

Le service LexFCS officiel concerne les lemmes. Les autres usages passent par les données officiellement publiées ou par un accord avec le projet TLA. Le manifeste du connecteur indique donc précisément quelles capacités sont réellement disponibles :

```
{
  "connector_id": "org.loracrafft.tla",
  "connector_version": "1.0.0",
  "provider_id": "de.bbaw.tla",
  "online": true,
  "capabilities": [
    "lemma_search",
    "lemma_lookup",
    "published_dataset_search"
  ],
  "unsupported_capabilities": [
    "image_recognition",
    "unrestricted_bulk_download"
  ]
}
```

11.13. Exports

Les exports sont eux-mêmes des traitements :

```
POST /internal/v1/exports
GET /internal/v1/exports/{export_id}
GET /internal/v1/exports/{export_id}/manifest
```

La demande précise révision, format, collections, profil éditorial et politique de droits. L'export retourne un manifeste avec empreintes de tous les fichiers produits.

```
{
  "document_id": "urn:lora:document:d-1",
  "revision_id": "urn:lora:revision:r-9",
  "format": "docx",
  "profile": "scientific-report-fr",
  "include": [
    "sources",
    "layout_objects",
    "glyph_hypotheses",
    "human_validations",
    "corpus_matches",
    "history"
  ]
}
```

Un export ne peut pas assouplir silencieusement les droits de ses dépendances. S'il omet une ressource restreinte, le manifeste le signale.

11.14. Modèle d'erreur

Toutes les interfaces utilisent une erreur structurée :

```
{
  "error": {
    "code": "UNSUPPORTED_IMAGE_SPACE",
    "message": "La géométrie ne référence pas l'image attendue.",
    "details": {
      "object_id": "urn:lora:layout-object:lo-42",
      "expected_space": "urn:lora:image:i-normalized"
    },
    "retryable": false,
    "request_id": "urn:lora:request:r-44",
    "correlation_id": "urn:lora:correlation:c-18"
  }
}
```

Les codes sont stables et non traduits. Les messages peuvent être localisés. details ne contient pas de secret ni de chemin privé inutile.

Codes minimaux :

Code	Signification	Nouvelle tentative
INVALID_REQUEST	requête incomplète ou incohérente	non sans correction
SCHEMA_VERSION_UNSUPPORTED	version de contrat inconnue	non avec le même client
UNKNOWN_OBJECT	identifiant absent de la révision	non
REVISION_CONFLICT	révision courante modifiée concurremment	oui après relecture
IDEMPOTENCY_CONFLICT	même clé, contenu	non sans nouvelle clé

Code	Signification	Nouvelle tentative
	différent	
INVALID_GEOMETRY	région vide, hors image ou incohérente	non sans correction
UNSUPPORTED_IMAGE_SPACE	espace de coordonnées incompatible	non sans conversion
ENGINE_UNAVAILABLE	moteur demandé indisponible	éventuellement
CORPUS_UNAVAILABLE	corpus distant indisponible	oui
LICENSE_RESTRICTED	opération interdite par les droits	non
RATE_LIMITED	quota temporairement dépassé	oui après délai
CANCELLED	traitement annulé	oui par nouveau traitement
INTERNAL_ERROR	erreur technique non classée	selon retryable

Pour HTTP, les codes peuvent être associés à 400, 401, 403, 404, 409, 422, 429, 500 ou 503. Le code métier reste toutefois la référence, y compris pour les appels Python et la CLI.

11.15. Événements

Les événements possèdent toujours :

- event_id ;
- event_type ;
- occurred_at ;
- document_id ;
- revision_id ;
- activity_id lorsque applicable ;
- correlation_id ;
- payload.

Exemple :

```
{
  "event_id": "urn:lora:event:e-77",
  "event_type": "glyph.hypothesis.created",
  "occurred_at": "2026-08-17T10:03:21Z",
  "document_id": "urn:lora:document:d-1",
  "revision_id": "urn:lora:revision:r-8",
  "activity_id": "urn:lora:activity:a-91",
  "correlation_id": "urn:lora:correlation:c-18",
  "payload": {
    "hypothesis_id": "urn:lora:glyph-hypothesis:gh-42"
```

```
}  
}
```

La livraison est au minimum « au moins une fois » : un consommateur peut recevoir le même événement plusieurs fois et mémorise donc les `event_id` déjà traités. L'ordre global de tous les événements n'est pas supposé. L'ordre d'un même document est reconstruit à partir des révisions et des dates, avec résolution explicite des conflits.

Les implémentations possibles sont un itérateur Python, un journal JSONL, une file de messages ou un flux HTTP de type SSE. Elles conservent la même enveloppe.

11.16. Authentification, autorisation et droits

Une installation monoposte peut utiliser l'identité locale configurée. Une installation partagée authentifie personnes, services et organisations.

Les autorisations minimales distinguent :

- lecture du document ;
- accès aux images originales ;
- ajout d'hypothèses ;
- création de validations ;
- arbitrage ;
- gestion de corpus ;
- export ;
- publication ;
- administration technique.

L'autorisation s'applique au document, à l'artefact et éventuellement à la région. Une URL signée ou un jeton d'accès ne doit pas être conservé comme provenance durable. La provenance conserve l'identité logique de la ressource et l'agent responsable, non le secret ayant permis l'accès.

Une requête vers un corpus externe transmet le minimum nécessaire. Une inscription inédite ou restreinte ne doit jamais être envoyée à un service distant sans politique explicite.

11.17. Versionnement et compatibilité

Le chemin `/internal/v1` désigne la version majeure du contrat HTTP. Les objets continuent à déclarer leur propre `$schema`. Une évolution compatible ajoute des champs facultatifs ou de nouvelles capacités. Une évolution incompatible change la version majeure et fournit une migration.

La règle Morandi s'applique : une nouvelle famille de contrat commence à `1.0`, jamais à `0.x`.

Un serveur annonce :

- versions de contrats acceptées ;

- version préférée ;
- versions dépréciées ;
- date de retrait prévue ;
- lien vers le guide de migration.

Un client ne doit pas déduire la compatibilité d'un simple numéro de version logicielle. La matrice de compatibilité relie explicitement noyau, schémas, modules, moteurs, catalogues et connecteurs.

11.18. Observabilité et audit

Chaque requête technique produit des journaux structurés avec `request_id`, `correlation_id`, module, durée, statut et volumes. Les métriques comprennent :

- temps d'attente et d'exécution ;
- nombre d'objets traités ;
- réussites, rejets honnêtes et erreurs ;
- usage mémoire et, si utile, GPU ;
- temps passé dans les services externes ;
- cache utilisé ou non ;
- version des dépendances.

Ces métriques servent au diagnostic. Elles ne remplacent pas les objets scientifiques de provenance. Un journal peut être purgé selon la politique d'exploitation ; les activités nécessaires à la reproduction restent dans le LoraDocument.

11.19. Tests de conformité des API

Toute implémentation d'API doit être soumise à une suite commune couvrant :

- validation des schémas d'entrée et de sortie ;
- appel répété avec la même clé d'idempotence ;
- conflit de révision ;
- pagination stable ;
- résultat vide légitime ;
- erreur locale et résultat partiel ;
- annulation et reprise ;
- événement livré deux fois ;
- version de contrat non supportée ;
- artefact inaccessible ;
- corpus externe indisponible ;
- refus pour licence ou autorisation ;
- absence de mutation des objets d'entrée ;
- conservation de la provenance ;
- fonctionnement local sans réseau pour les capacités déclarées offline.

La conformité technique ne suffit pas à certifier la qualité d'un moteur. Elle garantit seulement qu'un moteur, bon ou mauvais, expose honnêtement ses entrées, ses sorties, ses erreurs et sa provenance.

12. Formats JSON

12.1. Règles générales

- encodage UTF-8 ;
- dates ISO 8601 en UTC ;
- nombres décimaux JSON, sans NaN ni infini ;
- identifiants stables sous forme de chaînes ;
- propriétés en snake_case ;
- tableaux ordonnés uniquement lorsque l'ordre est sémantique ;
- champ \$schema dans les documents échangeables ;
- extension contrôlée via extensions.

Un champ absent signifie « non fourni ». null signifie « explicitement sans valeur » uniquement si le schéma l'autorise. Une liste vide signifie « recherche effectuée, aucun élément » lorsque l'activité correspondante est présente.

12.2. Compatibilité avec le prototype

Le format actuel de LoraLayout :

```
{
  "generated_at": "...",
  "source_image": "test_2.png",
  "layout": {
    "orientation": "vertical",
    "bands": [],
    "cadrats": []
  }
}
```

est accepté par un adaptateur d'import. LoraGlyph enrichit historiquement chaque cadrat avec id, classification, candidates, status, classifier_details et crop_path. Depuis l'implémentation de LoraDocument 1.0, un second adaptateur migre effectivement ces valeurs vers des GlyphHypothesis séparées et transforme les anciennes validations incorporées en HumanValidation. La projection historique reste produite pour audit et compatibilité, mais elle n'est plus le stockage scientifique de référence.

12.3. Objet géométrique

```
{
  "id": "urn:lora:layout-object:lo-1",
  "object_type": "QUADRAT",
  "geometry": {
    "space": "urn:lora:image:i-normalized",
    "shape": "bbox",
    "x": 46,
    "y": 18,
    "width": 38,
```

```

    "height": 37,
    "unit": "pixel"
  },
  "relations": [
    {"type": "part_of", "target_id": "urn:lora:layout-object:column-1"}
  ]
}

```

12.4. Groupe MdC aligné

```

{
  "id": "urn:lora:mdc:m-1",
  "reading_order_id": "urn:lora:reading-order:r-1",
  "root": {
    "type": "horizontal_group",
    "children": [
      {
        "type": "sign",
        "code": "G17",
        "glyph_refs": ["urn:lora:layout-object:lo-1"]
      },
      {
        "type": "vertical_group",
        "children": [
          {
            "type": "sign",
            "code": "X1",
            "glyph_refs": ["urn:lora:layout-object:lo-2"]
          },
          {
            "type": "unknown",
            "glyph_refs": ["urn:lora:layout-object:lo-3"]
          }
        ]
      }
    ]
  },
  "serialization": "G17-X1:?",
  "status": "proposed"
}

```

12.5. Analyse linguistique

```

{
  "id": "urn:lora:analysis:la-1",
  "mdc_id": "urn:lora:mdc:m-1",
  "transliteration": "m ...",
  "tokens": [
    {
      "id": "t1",
      "form": "m",
      "lemma": "m",
      "part_of_speech": "PREP",
      "gloss": "dans, avec, comme",
      "source_node_refs": ["urn:lora:mdc-node:n1"],

```

```
    "analyses": [  
      {"sense": "spatial", "score": 0.55},  
      {"sense": "instrumental", "score": 0.45}  
    ]  
  }  
]
```

12.6. Évolution des schémas

Les ajouts facultatifs sont compatibles. Renommage, changement de type ou de sémantique nécessitent une nouvelle version majeure de schéma. Les migrations sont des activités traçables qui conservent le document source.

13. Charte de nommage

13.1. Portée normative

Cette charte s'applique au code, aux objets scientifiques, aux schémas, aux API et aux fichiers produits par Loracrafft. Son objectif est de permettre à un développeur d'identifier immédiatement la nature d'un élément et d'éviter que deux modules emploient des noms différents pour le même concept.

Une dérogation est possible pour un standard externe ou une contrainte de compatibilité. Elle doit être documentée dans l'adaptateur ; elle ne modifie pas le vocabulaire interne.

13.2. Noms des modules

Les modules officiels utilisent UpperCamelCase, commencent par Lora et portent un nom de responsabilité :

LoraCapture
LoraImage
LoraLayout
LoraDirection
LoraGlyph
LoraReview
LoraMdC
LoraGrammar
LoraLexicon
LoraCorpus
LoraReasoner
LoraTranslate
LoraSpeech
LoraWorkflow

Règles :

- le préfixe Lora est écrit avec une majuscule initiale, sans trait d'union ;
- les acronymes consacrés suivent leur graphie officielle : MdC, SDK, API ;
- un nom décrit une responsabilité, non une technologie : LoraGlyph, pas LoraCNN ;
- le nom d'un module ne contient pas sa version ;
- une implémentation expérimentale non intégrée au noyau n'utilise pas seule l'espace de noms officiel.

Les paquets Python utilisent lowercase sans préfixe redondant lorsque le paquet parent fournit déjà l'espace de noms :

loracrafft.capture
loracrafft.glyph
loracrafft.reasoner
lorasdk

13.3. Noms des types d'objets

Les noms de types utilisent UpperCamelCase et un suffixe qui indique leur rôle.

Suffixe	Usage	Exemple
Document	agrégat documentaire principal	LoraDocument
Source	témoin ou ressource d'origine	ImageSource
Artifact	fichier ou produit numérique	ImageArtifact
Object	élément observé ou structurel	LayoutObject
Hypothesis	proposition révisable	GlyphHypothesis
Assertion	proposition du graphe de raisonnement	ReadingAssertion
Validation	décision humaine qualifiée	HumanValidation
Analysis	résultat analytique structuré	LinguisticAnalysis
Encoding	représentation codée	MdCEncoding
Match	rapprochement avec une ressource	CorpusMatch
Translation	proposition ou édition traduite	Translation
Activity	opération de provenance	GlyphRecognitionActivity
Agent	personne, logiciel ou organisation	SoftwareAgent
Event	fait du journal append-only	ValidationAddedEvent
Request	entrée d'une opération	GlyphRecognitionRequest
Result	sortie d'une opération	GlyphRecognitionResult
Error	exception ou erreur typée du SDK	SchemaValidationError
Diagnostic	anomalie non nécessairement fatale	LayoutDiagnostic
Manifest	description statique d'un paquet	CorpusManifest

Le suffixe Object n'est pas utilisé comme terme générique pour toute donnée. Il est réservé aux éléments observés ou structurels qui possèdent une identité scientifique, par exemple LayoutObject. Une proposition sur cet objet est une Hypothesis, et l'opération qui la produit est une Activity.

13.4. Noms des collections

Les collections JSON utilisent le pluriel en snake_case :

```
sources
images
layout_objects
reading_orders
glyph_hypotheses
human_validations
mdc_encodings
linguistic_analyses
lexical_links
corpus_matches
reasoning_objects
translations
speech_outputs
activities
history
```

Le nom de collection décrit le contenu, jamais le module qui l'a produit. On écrit `glyph_hypotheses`, pas `loraglyph_results`. La provenance indique le module producteur.

Une collection ne mélange pas des types de niveaux épistémiques différents. Les validations ne sont donc pas incorporées directement dans les hypothèses, même si une projection d'interface peut afficher ensemble leurs valeurs.

13.5. Noms des propriétés JSON

Les propriétés utilisent snake_case, des mots anglais et des noms explicites :

```
{
  "document_id": "...",
  "created_at": "...",
  "created_by": "...",
  "source_refs": [],
  "reading_order_id": "...",
  "score_kind": "engine_confidence"
}
```

Conventions :

- suffixe `_id` pour une référence unique ;
- suffixe `_ids` uniquement pour une liste d'identifiants homogènes ;
- suffixe `_ref` ou `_refs` lorsqu'une référence peut être un objet qualifié ou une URI externe ;
- suffixe `_at` pour un instant ISO 8601 ;
- suffixe `_date` pour une date civile sans heure ;
- suffixe `_sha256` pour une empreinte SHA-256 ;
- préfixe `is_`, `has_` ou `supports_` pour un booléen ;

- suffixe `_count` pour un entier de comptage ;
- suffixe `_version` pour une version déclarée ;
- suffixe `_uri` uniquement pour une URI, jamais pour un chemin local.

Les abréviations ambiguës sont évitées. `width` et `height` sont préférés à `w` et `h` dans le modèle cible. Les adaptateurs historiques peuvent accepter `x`, `y`, `w`, `h`.

Un nom ne répète pas inutilement son contexte. Dans un `GlyphCandidate`, `sign_ref` est préférable à `glyph_candidate_sign_reference`.

13.6. Valeurs d'énumération

Les valeurs techniques fermées utilisent `UPPER_SNAKE_CASE` lorsqu'elles représentent une catégorie :

GLYPH
GROUP
FRAGMENT
NOISE
UNKNOWN

Les états de cycle de vie utilisent `lower_snake_case` :

needs_review
in_progress
accepted
superseded

Cette distinction reprend l'existant de `LoraGlyph` et permet de distinguer visuellement une classe métier d'un état. Chaque énumération appartient à un schéma versionné ; une valeur inconnue ne doit pas être remplacée silencieusement par `UNKNOWN` si cela ferait perdre l'information originale.

13.7. URI

Les identifiants internes prennent la forme :

`urn:lora:<type>:<identifiant>`

Exemples :

`urn:lora:document:01K0ABC...`
`urn:lora:revision:01K0DEF...`
`urn:lora:image:01K0GHI...`
`urn:lora:layout-object:01K0JKL...`
`urn:lora:glyph-hypothesis:01K0MNO...`
`urn:lora:validation:01K0PQR...`
`urn:lora:activity:01K0STU...`

Le segment `<type>` utilise `lowercase-kebab-case`. Il est choisi dans un registre central. L'identifiant final est opaque : un client ne déduit pas l'auteur, la date, le parent ou le numéro de colonne depuis sa valeur.

Les ressources publiques peuvent recevoir une URI HTTPS résoluble :

<https://loracrafft.org/id/document/01K0ABC...>

La résolution peut rediriger vers une représentation humaine ou machine selon la négociation de contenu. L'URI publique et l'URN interne sont reliées explicitement ; elles ne sont pas reconstruites l'une depuis l'autre par convention implicite.

Les autorités externes utilisent leur propre espace :

gardiner:G17
unicode:U+13153
iiif:[https://example.org/iiif/...](https://example.org/iiif/)

Un préfixe compact n'est employé que si son expansion est déclarée dans le contexte du document.

13.8. UUID, ULID et identifiants locaux

L'identifiant recommandé pour les nouveaux objets est un **ULID** ou un **UUIDv7**, afin de combiner unicité distribuée et ordre temporel approximatif. Le choix définitif doit être unique dans le noyau ; tant qu'il ne l'est pas, le SDK accepte les deux et les traite comme valeurs opaques.

Règles :

- génération côté service qui crée l'objet ;
- jamais d'identifiant dérivé d'un index de tableau ;
- jamais de réutilisation après suppression logique ;
- pas d'information personnelle encodée ;
- casse et représentation normalisées par le SDK ;
- validation syntaxique sans inférence sémantique.

Les petits identifiants tels que t1 sont autorisés uniquement à l'intérieur d'un objet autonome lorsqu'ils ne sont jamais référencés de l'extérieur. Dès qu'un élément est une cible de provenance, de validation ou d'alignement inter-collection, il reçoit une URI stable.

13.9. Nommage des activités et événements

Une activité utilise un nom nominal décrivant l'opération :

ImageNormalizationActivity
LayoutDetectionActivity
GlyphRecognitionActivity
HumanReviewActivity
CorpusSearchActivity
TranslationGenerationActivity

Un type d'événement décrit un fait accompli au passé conceptuel et se termine par Event dans le code :

ImageNormalizedEvent
GlyphHypothesisCreatedEvent

ValidationAddedEvent
RevisionPublishedEvent

Sur le bus, les types utilisent une notation stable en minuscules :

image.normalized
glyph.hypothesis.created
validation.added
revision.published

Les commandes décrivent une intention et ne sont pas confondues avec les événements : RecognizeGlyphs n'est pas glyphs.recognized.

13.10. Nommage des moteurs, modèles et corpus

Un moteur possède :

- un engine_id global et stable, qualifié par l'organisation ;
- un nom humain ;
- une version logicielle ;
- éventuellement un model_id et une version de modèle distincts.

engine_id: org.loracrafft.template-recognizer
engine_version: 1.0.1
model_id: org.example.glyph-vit
model_version: 2026-07

Un changement de poids n'est pas masqué par une version inchangée. L'empreinte du modèle ou du catalogue complète la version.

Un corpus utilise un identifiant qualifié :

corpus_id: org.example.middle-kingdom-stelae
release_id: 2026.1

La version de publication du corpus est distincte de la version de son logiciel d'indexation.

13.11. Nommage des fichiers

Les fichiers contrôlés par Loracrafft utilisent lowercase-kebab-case lorsque leur nom est destiné à l'humain, et un identifiant stable lorsqu'ils sont adressés par le système.

Exemples :

extension-manifest.json
corpus-manifest.json
loradocument.schema.json
glyph-recognition-report.json
revision-01K0ABC-manifest.json

Les images originales conservent leur nom source dans les métadonnées. Le stockage peut les renommer par empreinte, mais ne perd jamais original_filename.

Les suffixes recommandés sont :

*.schema.json	schéma JSON
*.manifest.json	manifeste
*.events.jsonl	journal append-only
*.report.json	rapport machine
*.md	documentation

13.12. Nommage dans le code

Pour Python :

- modules, fonctions et variables : snake_case ;
- classes et protocoles : UpperCamelCase ;
- constantes : UPPER_SNAKE_CASE ;
- membres internes : préfixe _ ;
- exceptions : suffixe Error ;
- protocoles : nom de capacité, sans préfixe I.

Exemples :

```
class GlyphRecognizer: ...
class CorpusProvider: ...
class SchemaValidationError(Exception): ...
```

```
DEFAULT_TOP_K = 3
```

```
def recognize_glyphs(request: GlyphRecognitionRequest): ...
```

Les noms data, item, object, result ou payload sont acceptables dans une portée très locale, mais pas comme noms de contrats publics lorsqu'un terme scientifique plus précis existe.

13.13. Compatibilité et dépréciation

Un nom public n'est pas renommé sans migration. La dépréciation suit trois étapes :

1. le nouveau nom est ajouté et l'ancien reste accepté ;
2. le SDK émet un diagnostic de dépréciation ;
3. l'ancien nom est retiré dans une version incompatible du schéma.

Les adaptateurs conservent la valeur originale afin de permettre un export fidèle. Un registre de décisions de nommage documente les termes réservés, synonymes rejetés et exceptions.

13.14. Résumé obligatoire pour les contributeurs

Avant d'introduire un nouveau nom, le contributeur vérifie :

1. le concept existe-t-il déjà ?
2. s'agit-il d'une observation, hypothèse, validation, analyse ou activité ?

3. le suffixe exprime-t-il correctement cette nature ?
 4. le nom décrit-il une responsabilité plutôt qu'une technologie ?
 5. la propriété JSON respecte-t-elle snake_case ?
 6. l'identifiant est-il stable et opaque ?
 7. le terme est-il cohérent avec les vocabulaires égyptologiques utilisés ?
 8. une migration est-elle nécessaire ?
-

14. Scénarios complets : de la photo à la traduction

14.1. Scénario A — inscription nette en colonnes

Import et préparation

Une photographie TIFF est importée. LoraCapture calcule son empreinte et enregistre la cote du monument. LoraImage corrige un angle de 1,8°, applique un contraste local modéré et produit une image de travail.

Géométrie

LoraLayout détecte cinq colonnes et 142 régions. L’overlay révèle deux séparateurs pris pour des cadrats ; un opérateur les requalifie. Un cartouche coupé par le bord est ajouté manuellement comme région ouverte.

Direction et glyphes

Les figures orientables indiquent une lecture de droite à gauche, colonnes parcourues de droite à gauche et signes de haut en bas. LoraGlyph classe 96 objets GLYPH, 18 GROUP, 9 FRAGMENT, 7 NOISE et 12 UNKNOWN. Le moteur de gabarits n’ayant pas de catalogue suffisant, plusieurs listes de candidats restent vides. Cette absence déclenche une revue au lieu d’une invention.

Encodage et langue

Les groupes sont segmentés, puis LoraMdC construit un arbre aligné. Une séquence admet deux lectures de déterminatif. LoraGrammar propose deux segmentations ; l’une produit une construction attestée, l’autre requiert une graphie rare.

Corpus et conclusion

LoraCorpus retrouve deux formules parallèles de même période. Le raisonneur préfère la première segmentation, tout en conservant la seconde. LoraTranslate produit une version littérale et une version fluide. Le dossier publié permet de cliquer sur chaque segment traduit pour afficher glyphes, validation et parallèles.

14.2. Scénario B — inscription endommagée

Une zone contient trois traits partiels. LoraLayout la marque DAMAGE et crée deux régions candidates. LoraGlyph les classe FRAGMENT. Aucun code Gardiner n’est proposé.

Un expert suggère que les fragments appartiennent à un seul signe, avec certitude faible. Cette décision crée une hypothèse de fusion. Le corpus retrouve une formule où un signe attendu occupe cette position, mais la forme ne permet pas de le confirmer.

Le dossier conserve :

- observation : trois fragments ;
- hypothèse : fragments d’un même signe ;

- restitution : signe attendu par parallèle ;
- décision éditoriale : restitution entre crochets ;
- traduction : mot restitué marqué comme tel.

La publication ne transforme donc pas la restitution en signe observé.

14.3. Scénario C — sens de lecture ambigu

Une scène symétrique présente deux personnages faisant face au centre. La géométrie autorise deux parcours. LoraDirection produit deux graphes. Les reconnaissances glyphiques sont communes, mais les séquences diffèrent.

L'analyse A forme un titre et une proposition cohérente. L'analyse B forme plusieurs lexèmes possibles mais échoue sur l'ordre syntaxique. Un parallèle iconographique soutient A. Le raisonneur classe A en tête sans supprimer B ; la publication affiche la direction choisie et son argument.

14.4. Scénario D — correction tardive

Après publication de travail, un chercheur requalifie un objet G43 en G17. L'événement de validation invalide un nœud MdC, deux tokens, une analyse et une phrase traduite. Les transformations d'image, régions et autres colonnes ne sont pas recalculées.

LoraWorkflow produit une nouvelle branche, exécute les modules nécessaires et génère une nouvelle révision. Une comparaison indique :

- signe modifié ;
 - translittération affectée ;
 - analyse grammaticale devenue préférable ;
 - traduction mise à jour ;
 - ancienne révision toujours citable.
-

15. Feuille de route de développement

15.1. Lecture immédiate

La feuille de route fonctionnelle est organisée en quatre phases :

Phase	Chaîne principale	Finalité
Phase 1	LoraLayout → LoraGlyph → LoraReview	Observer, segmenter, reconnaître et valider
Phase 2	LoraMdC → LoraGrammar → LoraLexicon	Encoder et analyser le texte
Phase 3	LoraCorpus → LoraReasoner	Comparer et argumenter
Phase 4	LoraTranslate → LoraSpeech	Traduire, publier et restituer

Au 17 août 2026, le projet se situe toujours en **Phase 1 expérimentale**, mais les fondations transversales ont progressé :

- LoraDocument 1.0, son schéma JSON, ses adaptateurs et ses validations multi-annotateurs sont implémentés ;
- LoraImage redresse automatiquement le cas réel incliné d'environ 44,7° et conserve sa provenance ;
- LoraLayout détecte correctement les deux registres de test_2, mais ne délimite pas encore toutes les unités hiéroglyphiques ;
- LoraGlyph 1.0.1 possède un pipeline et plusieurs voies expérimentales de comparaison morphologique ;
- une base locale de 6 722 fichiers glyphiques est disponible mais son manifeste et sa qualification restent à produire ;
- LoraReview possède le modèle de validation durable, mais pas encore l'interface visuelle de correction géométrique ;
- les phases 2 à 4 décrivent l'architecture cible et ne doivent pas être présentées comme implémentées.

15.2. Phase 1 — voir et valider

Modules

LoraLayout
↓
LoraGlyph
↓
LoraReview

Objectif

Transformer une image préparée en un ensemble d'objets visuels et d'hypothèses glyphiques validables, sans produire prématurément du texte ou une traduction.

Travaux

Pour LoraLayout :

- stabiliser le contrat de géométrie ;
- introduire polygones, espaces image et relations ;
- améliorer les cartouches ouverts et les groupes ;
- achever la vérité terrain de test_2 avant d'ouvrir test_3 ;
- figer la segmentation validée de test_2 comme référence de non-régression ;
- représenter explicitement les motifs NON_GLYPH et les cas OUTLIER ;
- constituer un corpus d'évaluation réel à partir des scans sans fuite documentaire ;
- mesurer précision et rappel par type d'objet.

Pour LoraGlyph :

- maintenir l'adaptateur désormais implémenté vers des GlyphHypothesis séparées ;
- produire le manifeste de la base Monfort et versionner catalogues et référentiels ;
- calibrer les scores ou les qualifier explicitement ;
- évaluer séparément moments de Hu, silhouettes, projections et contours avant toute fusion de scores ;
- préparer plusieurs moteurs comparables ;
- développer fusion de fragments et segmentation des groupes sans inventer de signe.

Pour LoraReview :

- construire en priorité l'interface visuelle de correction géométrique de test_2 ;
- ajouter fusion, scission, rejet, report et niveau de certitude ;
- exposer dans l'interface les validations multi-annotateurs déjà représentées par le noyau ;
- ajouter l'arbitrage sans modifier les décisions antérieures ;
- préparer la curation des exemples validés.

Livable

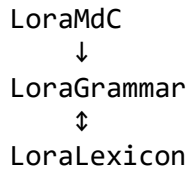
Une inscription peut être segmentée, reconnue partiellement et revue. Chaque objet visuel, candidat et validation possède une provenance. L'absence de reconnaissance est représentée honnêtement.

Critère de passage

La Phase 1 est achevée lorsqu'un jeu d'inscriptions réelles peut être traité de façon reproductible, que deux annotateurs ne s'écrasent pas mutuellement et que les résultats sont migrés dans un LoraDocument valide. Le deuxième et le troisième critères possèdent désormais une implémentation testée ; le premier reste bloqué par la segmentation exacte de test_2.

15.3. Phase 2 — encoder et analyser

Modules



Objectif

Transformer les glyphes validés et leur ordre de lecture en une représentation textuelle structurée, puis proposer des analyses linguistiques sourcées.

Travaux

Pour LoraMdC :

- définir l’AST interne ;
- construire l’alignement entre feuilles MdC et objets visuels ;
- gérer groupes, lacunes, restitutions et signes inconnus ;
- développer parseur, sérialiseur et rendu ;
- tester les conversions MdC et Unicode.

Pour LoraGrammar :

- définir tokens, lemmes, traits et relations syntaxiques ;
- créer les premières règles de segmentation et morphologie ;
- gérer les branches concurrentes ;
- produire des diagnostics explicables ;
- constituer un jeu d’or avec désaccords documentés.

Pour LoraLexicon :

- choisir et documenter les référentiels de signes ;
- structurer les entrées lexicales et leurs sources ;
- gérer graphies, valeurs, époques et noms propres ;
- clarifier les droits des ressources ;
- exposer une API de recherche versionnée.

Livrable

Une séquence de glyphes validée produit un encodage MdC aligné, une translittération et plusieurs analyses possibles. Chaque valeur lexicale et chaque règle importante renvoie à une source ou à une ressource versionnée.

Critère de passage

La Phase 2 est achevée lorsqu'une correction de glyphe invalide uniquement les analyses dépendantes et qu'un égyptologue peut suivre l'alignement continu entre image, MdC, translittération et analyse.

15.4. Phase 3 — comparer et raisonner

Modules

LoraCorpus



LoraReasoner

Objectif

Mettre les lectures en relation avec des témoins et connaissances externes, puis produire des conclusions argumentées qui conservent alternatives et contre-arguments.

Travaux

Pour LoraCorpus :

- implémenter le modèle œuvre-témoin-passage-édition ;
- définir les profils de normalisation ;
- créer des index graphique, MdC, lexical et syntaxique ;
- gérer lacunes, variantes et contraintes ;
- documenter droits, citations et dépendances éditoriales.

Pour LoraReasoner :

- définir le graphe d'assertions ;
- créer le registre des règles ;
- distinguer preuves, indices, contradictions et hypothèses nécessaires ;
- calculer des branches et contrefactuels ;
- produire des explications lisibles et vérifiables.

Livable

Une analyse peut retrouver des parallèles citables et construire plusieurs scénarios. Une conclusion préférée indique ce qui la soutient, ce qui la contredit et ce qui changerait si une hypothèse amont était remplacée.

Critère de passage

La Phase 3 est achevée lorsque les résultats de corpus sont reproductibles pour une version d'index et qu'aucune conclusion du raisonneur n'existe sans chaîne de dépendances jusqu'aux observations ou aux autorités documentaires.

15.5. Phase 4 — traduire et restituer

Modules

LoraTranslate



LoraSpeech

Objectif

Produire des traductions alignées, révisables et publiables, puis proposer lorsque cela est pertinent une restitution sonore clairement qualifiée.

Travaux

Pour LoraTranslate :

- créer les profils littéral, philologique et fluide ;
- aligner chaque segment sur l'analyse source ;
- préserver lacunes, restitutions et alternatives ;
- intégrer une revue humaine obligatoire ;
- produire exports scientifiques et révisions citables.

Pour LoraSpeech :

- définir les profils de lecture conventionnelle ou reconstruite ;
- aligner texte et audio ;
- documenter les limites de la vocalisation ;
- fournir un lecteur accessible ;
- appliquer les droits et politiques de sensibilité.

Livable

Une publication relie la traduction à la translittération, au MdC, aux validations et à l'image. Une sortie audio ne se présente jamais comme la prononciation certaine de l'égyptien ancien.

Critère d'achèvement

La Phase 4 est achevée lorsqu'une révision publiée est citable, exportable et entièrement traçable de la traduction jusqu'aux sources.

15.6. Fondations transversales

Certains composants ne constituent pas une phase autonome : ils rendent chaque phase possible.

Composant	Rôle dans la feuille de route
LoraCapture	import, intégrité, contexte et droits
LoraImage	préparation non destructive et espaces de coordonnées

Composant	Rôle dans la feuille de route
LoraDirection	ordre de lecture, requis avant la finalisation MdC
LoraDocument	dossier scientifique commun
LoraWorkflow	orchestration, reprise et invalidation
LoraSDK	contrats et outils destinés aux contributeurs

Le noyau LoraDocument, la provenance, la charte de nommage et les schémas JSON commencent pendant la Phase 1 et évoluent avec toutes les phases. LoraDirection doit être suffisamment mature avant la sortie de Phase 2.

15.7. Chantiers permanents

À chaque phase :

- sécurité, droits et licences ;
- documentation des API et du SDK ;
- migrations de schéma ;
- tests unitaires, d'intégration et scientifiques ;
- reproductibilité et provenance ;
- mesure des biais et du domaine de validité ;
- sauvegarde et archivage ;
- accessibilité et internationalisation.

15.8. Règle de priorité

La priorité va au modèle de données, à la revue et à la provenance avant l'ajout de moteurs plus puissants. Un excellent reconnaisseur branché sur un dossier non traçable produirait des résultats difficiles à exploiter scientifiquement ; un socle traçable permet au contraire d'améliorer progressivement tous les moteurs.

16. Conclusion

Loracrafft est une plateforme scientifique de lecture, d'interprétation et de traduction des inscriptions égyptiennes. Son architecture repose sur une idée simple mais exigeante : une image, une détection, une identification, une analyse et une traduction ne sont pas des versions successives d'une même vérité, mais des objets de nature différente reliés par des preuves.

En conservant séparément observations, hypothèses, validations, raisonnements et décisions éditoriales, Loracrafft peut intégrer des moteurs de plus en plus performants sans sacrifier l'explicabilité. Il devient alors possible de produire non seulement une traduction, mais un dossier scientifique reproductible, révisable et durable.

Annexes

Annexe A — Catégories de Gardiner

La liste de Gardiner organise les signes en catégories thématiques. Le tableau ci-dessous sert de repère ; la source de référence et l'édition utilisées devront être déclarées dans LoraLexicon.

Code	Catégorie générale
A	Homme et occupations
B	Femme et occupations
C	Divinités anthropomorphes
D	Parties du corps humain
E	Mammifères
F	Parties de mammifères
G	Oiseaux
H	Parties d'oiseaux
I	Amphibiens et reptiles
K	Poissons et parties de poissons
L	Invertébrés et petits animaux
M	Arbres et plantes
N	Ciel, terre et eau
O	Bâtiments et parties de bâtiments
P	Bateaux et parties de bateaux
Q	Mobilier domestique et funéraire
R	Mobilier cultuel et emblèmes sacrés
S	Couronnes, vêtements et accessoires
T	Guerre, chasse et boucherie
U	Agriculture, artisanat et métiers
V	Cordes, fibres, paniers et sacs
W	Vases de pierre et poterie
X	Pains et gâteaux
Y	Écriture, jeux et musique
Z	Traits, signes géométriques et dérivés
Aa	Signes non classés

Les codes étendus, variantes et suffixes doivent préciser leur autorité. G17 signifie une entrée de référentiel, non une interprétation phonétique unique.

Annexe B — conventions MdC minimales

Le Manuel de Codage exprime à la fois les signes et leur groupement. Les conventions exactes dépendent du dialecte de MdC et du moteur de rendu. Le profil Loracrafft doit donc publier une grammaire formelle.

Principes de travail :

- - : juxtaposition dans le flux principal ;
- : : groupement vertical ;
- * : groupement horizontal serré selon le profil ;
- parenthèses : portée explicite d'un groupe ;
- codes Gardiner : identifiants de signes ;
- marqueur dédié : objet inconnu ou lacune ;
- annotations structurelles : dommage, restitution, incertitude.

La chaîne n'est jamais l'unique représentation. L'AST MdC interne demeure la source d'alignement et permet plusieurs sérialisations.

Annexe C — conventions éditoriales

Le profil de publication doit définir sans ambiguïté :

- signes observés mais non identifiés ;
- signes probables ;
- lacunes ;
- restitutions ;
- ajouts de l'éditeur ;
- suppressions proposées ;
- passages illisibles ;
- inversions honorifiques ;
- graphies fautives ou atypiques ;
- changements de main ou de phase.

Ces conventions sont portées par des objets structurés avant d'être rendues sous forme de crochets, points ou signes typographiques.

Annexe D — statuts recommandés

Domaine	Valeurs principales
Objet visuel	detected, reviewed, rejected, superseded
Classification	GLYPH, GROUP, FRAGMENT, NOISE, UNKNOWN ; diagnostics candidats NON_GLYPH, OUTLIER
Hypothèse	proposed, accepted, rejected, deferred, undecidable
Traitement	queued, running, succeeded, partial, failed, cancelled
Révision	working, reviewed, publishable, published, superseded

Les valeurs sont fermées dans un schéma donné. Les libellés d’interface peuvent être traduits.

Annexe E — conventions d’identifiants

Exemples :

```
urn:lora:document:...  
urn:lora:revision:...  
urn:lora:image:...  
urn:lora:layout-object:...  
urn:lora:glyph-hypothesis:...  
urn:lora:validation:...  
urn:lora:activity:...
```

Les identifiants sont opaques. Le type apparent dans l’URN facilite le diagnostic mais ne doit pas être la seule source de typage.

Annexe F — niveaux de confiance humaine

Un vocabulaire simple peut être utilisé :

- certain : aucune alternative raisonnable dans le cadre considéré ;
- high : alternative très peu probable ;
- medium : lecture préférée mais concurrence réelle ;
- low : proposition fragile ;
- undecidable : données insuffisantes.

Chaque projet doit accompagner ces mots d'instructions et d'exemples pour améliorer l'accord inter-annotateurs.

Annexe G — tests scientifiques minimaux

Une livraison de module doit comprendre :

- tests de schéma ;
- tests de non-modification des entrées ;
- tests de reproductibilité ;
- cas sans résultat ;
- cas ambigus ;
- cas endommagés ;
- cas hors domaine ;
- comparaison à un corpus annoté ;
- rapport d'erreurs ventilé ;
- vérification des dépendances et de la provenance.

Pour LoraGlyph, les tests existants couvrent déjà plusieurs invariants importants : bornage des vignettes, exclusion des fragments et groupes du reconnaisseur, conservation du JSON source, classement des candidats, interdiction d'écraser l'entrée et historique des corrections.

La campagne actuelle ajoute une hiérarchie obligatoire de non-régression :

Niveau	Cas	Critère de passage
0	glyphe G1 isolé et propre	une image, un seul cadrat, un seul objet reconnaissable
1	test_1	conserver les détections déjà validées lors de toute modification de segmentation
2	test_2	deux registres corrects et chaque unité hiéroglyphique correctement délimitée
3	test_3 et suivants	interdits tant que le niveau 2 n'est pas validé à 100 %

Le « 100 % » du niveau 2 ne désigne pas un score moyen ni un bon aspect visuel global. Il signifie que toutes les unités attendues sont présentes, qu'aucun cadre ne fusionne indûment deux unités et qu'aucun faux cadre n'est accepté comme signe.

Chaque exécution de test produit un dossier output\<<version>\<test>\ et un rapport contenant :

- fichier source et empreinte ;
- version du programme et du catalogue ;
- dates et heures de début et de fin ;
- durée ;
- paramètres de redressement et de segmentation ;
- nombres de régions par classe et, pour les overlays historiques, par couleur ;
- codes Gardiner proposés, noms lorsque disponibles, scores et type de score ;

- avertissements, erreurs et objets sans candidat.

Les sorties d'une version ne sont jamais écrasées par la suivante. Les images attendues, annotations et rapports validés constituent la référence de comparaison automatique.

Annexe H — Glossaire

Assertion — proposition explicite susceptible d’être soutenue ou contredite.

Cadrat — unité graphique d’organisation spatiale d’un ou plusieurs signes.

Candidat — identification proposée par un moteur, non validée par défaut.

Conclusion — assertion résultant d’un raisonnement documenté.

Entité — donnée ou artefact dans le modèle de provenance.

Hypothèse — proposition interprétative dont le statut reste révisable.

MdC — Manuel de Codage, convention de représentation des hiéroglyphes et de leurs groupements.

Observation — description issue directement d’une source ou d’une mesure.

Provenance — histoire des agents, activités, sources et transformations d’un objet.

Révision — vue cohérente et identifiable d’un LoraDocument.

Témoin — occurrence matérielle particulière d’un texte.

Annexe I — Journal expérimental du 15 au 17 août 2026

Ce journal résume les changements qui ont une portée architecturale. Les numéros désignent des livraisons expérimentales ; ils ne certifient pas un niveau de qualité scientifique.

Version ou groupe	Résultat principal	Conséquence architecturale
1.0.1	Réunion de LoraLayout et LoraGlyph 1.0.1 autour de LoraDocument 1.0. Sur l'ancien exemple vertical LoraLayout/exemples/test_2.png : 172 objets géométriques, 165 hypothèses et 165 vignettes, dont 121 GLYPH, 35 GROUP et 9 FRAGMENT. Aucun code Gardiner inventé en l'absence de gabarits.	Le document scientifique commun devient le stockage de référence ; les JSON historiques restent des projections d'audit.
1.0.2	Détection Hough de l'orientation dominante, calcul circulaire modulo 90°, correction du sens de rotation et traçage de l'angle. Environ 44,7° mesurés sur la stèle réelle.	Le redressement doit précéder toute géométrie ; chaque région déclare l'espace image auquel elle appartient.
1.0.3	Redressement confirmé sur l'image réelle ; apparition nette du problème composante-versus-signe.	La qualité du cadrage devient une précondition de la reconnaissance.
1.0.4	Fusion expérimentale de 173 composantes brutes en 56 groupes ; cas de sous-fusion, sur-fusion et motifs étrangers au texte.	Introduire groupes, fragments, diagnostics NON_GLYPH/OUTLIER et revue géométrique.
1.0.5	Échec sur une inscription à deux lignes : les cadrats attendus ne sont pas produits.	Abandon du découpage global avant compréhension de la structure.
1.0.6	Mise en œuvre de la stratégie registres d'abord.	Ordre imposé : surface → registres/lignes/colonnes → cadrats → signes.
1.0.7 à 1.0.9	Itérations sur test_1 et test_2, remise en cause de	Les couleurs d'overlay sont des diagnostics d'interface,

Version ou groupe	Résultat principal	Conséquence architecturale
	certaines cadres colorés et décompression de la base de glyphes.	jamais des types scientifiques ; le catalogue est une ressource versionnée séparée.
1.0.10	Stratégie Gardiner-first pour conserver, fusionner ou scinder les régions ; ajout des horodatages et de la durée ; 27 tests techniques annoncés.	Le rapport d'exécution devient un artefact obligatoire et test_1 est protégé contre les régressions.
1.0.11	Résumé final enrichi avec nom de fichier, version, codes Gardiner et scores.	Toute décision automatique doit être lisible et auditable sans ouvrir les journaux internes.
1.0.12	Organisation des sorties dans output\<version>\<test>\, sans écrasement ; gel de test_2. Un essai G1 autour de 0,748 met en évidence la faiblesse d'un score de gabarit brut.	Les régressions deviennent comparables ; aucun seuil unique ne vaut validation scientifique.
1.0.13-enhanced	Ajout du G1 isolé comme niveau 0 de non-régression ; essais renforcés sur morphologie et cartouches.	Règle minimale : une image propre contenant un glyphe isolé doit produire exactement un carrat avant toute reconnaissance.

Les valeurs ci-dessus doivent rester reliées aux archives d'exécution correspondantes. Si une archive manque ou si un résultat est recalculé, le nouveau résultat reçoit une nouvelle activité et ne remplace pas rétrospectivement le journal.

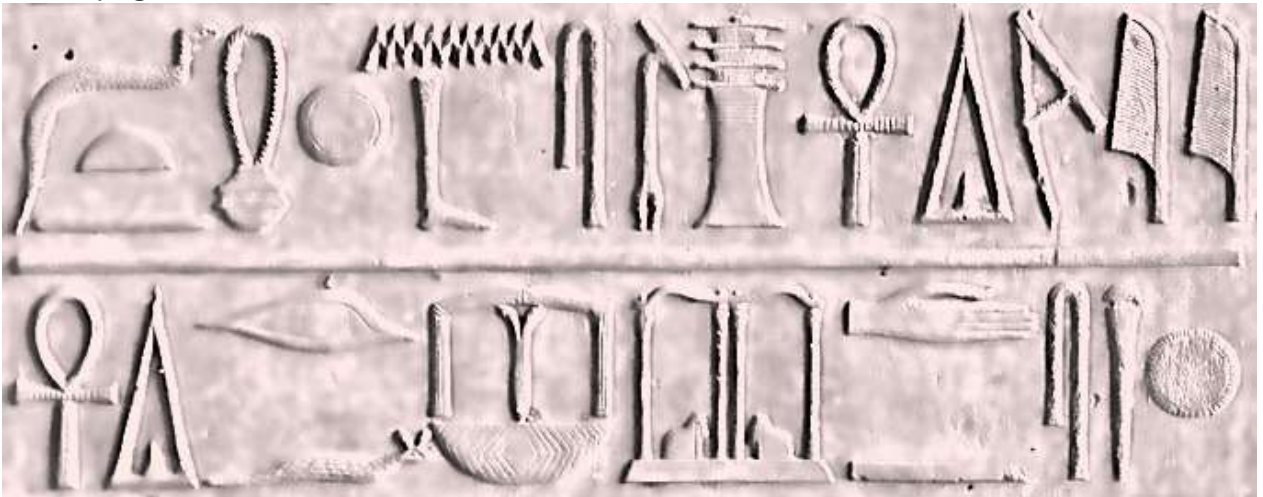
L'ancien fichier vertical nommé test_2.png dans le prototype initial et le test_2 à deux registres utilisé ensuite pour la campagne de segmentation sont deux témoins différents malgré leur nom historique identique. Les manifestes futurs leur attribueront des identifiants et des empreintes distincts ; un nom de fichier seul ne doit jamais identifier un cas de test.

Annexe J - Images de tests

test_1.png



test_2.png



test_3.png





test_5.png



Contributeurs

Didier Morandi : Concepteur du projet, architecture fonctionnelle, expression du besoin, validation scientifique et contrôle qualité.

OpenAI ChatGPT 5.6 : Assistance à la conception de l'architecture, réflexion technique, rédaction de spécifications et production d'exemples de code.

OpenAI Codex : Développement du logiciel, prototypage, tests unitaires et assistance au débogage.

Anthropic Claude.AI : Assistance à la rédaction de la documentation technique, des manuels utilisateur et des pages Web.