Compétences Claude_QGIS

RAG de toutes les expériences de Claude dans l'utilisation de QGIS par MCP

- REX 06079
- Expertises Topologie - Claude et Qgis

REX 06079

Contexte

Affichage des IRIS de Mandelieu-la-Napoule (06079) depuis PostGIS Alteris, avec population graduée, étiquettes, fond orthophoto IGN Géoplateforme et cadastre vecteur WFS — entièrement piloté depuis Claude Code via le MCP QGIS, sans interaction manuelle dans QGIS.

Données utilisées

Table / Source	Schéma / Provider	Contenu
`contours_iris_2025`	`insee_raw` (PostGIS Alteris)	Géométries IRIS France entière
`iris_2025_population`	`insee_raw` (PostGIS Alteris)	Population 2022 par IRIS (recensement)
Orthophoto IGN	Géoplateforme (WMTS → XYZ)	`HR.ORTHOIMAGERY.ORTHOPHOTOS` — sans clé API
Cadastre parcelles	Géoplateforme WFS `CADASTRALPARCELS.PARCELLAIRE_EXPRESS:parcelle`	Parcelles cadastrales vecteur — OGR driver

Colonnes clés — pièges à retenir

`contours_iris_2025` : colonnes standard sans espace — `code_insee`, `code_iris`, `nom_iris`, `type_iris`, `geom`

`iris_2025_population` : colonnes avec **espace traînant** (import INSEE brut) :

- Clé IRIS : `"IRIS "` (avec espace)
- Commune : `"COM "` (avec espace)
- Population totale 2022 : `"P22_POP "` (avec espace)
- Toutes les colonnes sont de type `text` — caster en `::numeric` avant usage

---

Résultats

9 IRIS pour 06079, population 2022 :

code_iris	nom_iris	population
060790101	Zone d'activités	24
060790102	IRIS 2	2 701
060790103	IRIS 3	2 000
060790104	IRIS 4	3 058
060790105	IRIS 5	2 197
060790106	IRIS 6	2 432
060790107	IRIS 7	3 188
060790108	IRIS 8	2 944
060790109	IRIS 9	2 656

Population min : **24 hab.** (zone d'activités/naturelle) — max : **3 188 hab.**

Cadastre : **26 439 parcelles** après reprojection EPSG:4326 → EPSG:3857 (COUNT=2000 limite la requête WFS initiale à 2000 features, mais la reprojection en mémoire conserve toutes les géométries valides).

Procédure complète

Prérequis

Tunnel SSH paramiko actif (voir page 229 — Option 2). À relancer à chaque redémarrage de QGIS :

```
import sys
sys.path.insert(0, r'C:\Users\eliob\AppData\Roaming\Python\Python312\site-packages')
import paramiko
import socket, threading, select

class SSHTunnel(threading.Thread):
    def __init__(self, ssh_host, ssh_user, ssh_password,
```

```
                remote_port, local_port=5433, remote_host='127.0.0.1'):
    super().__init__(daemon=True)
    self.ssh_host = ssh_host; self.ssh_user = ssh_user
    self.ssh_password = ssh_password; self.remote_host = remote_host
    self.remote_port = remote_port; self.local_port = local_port
    self._stop = threading.Event(); self.transport = None; self.server_sock = None

def run(self):
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(self.ssh_host, username=self.ssh_user, password=self.ssh_password)
    self.transport = client.get_transport()
    self.server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.server_sock.bind(('127.0.0.1', self.local_port))
    self.server_sock.listen(5); self.server_sock.settimeout(1.0)
    while not self._stop.is_set():
        try:
            conn, _ = self.server_sock.accept()
            threading.Thread(target=self._forward, args=(conn,), daemon=True).start()
        except socket.timeout:
            continue
    self.server_sock.close(); client.close()

def _forward(self, local_conn):
    try:
        chan = self.transport.open_channel('direct-tcpip',
            (self.remote_host, self.remote_port), local_conn.getpeername())
    except Exception:
        local_conn.close(); return
    while True:
        r, _, _ = select.select([local_conn, chan], [], [], 5)
        if local_conn in r:
            data = local_conn.recv(4096)
            if not data: break
            chan.send(data)
        if chan in r:
            data = chan.recv(4096)
            if not data: break
```

```
                local_conn.send(data)
        chan.close(); local_conn.close()

    def stop(self):
        self._stop.set()

tunnel = SSHTunnel('79.137.14.202', 'debian', 'RAW+NEXTE!', remote_port=5432, local_port=5433)
tunnel.start()
```

1. Charger l'orthophoto IGN (fond de carte)

Bug QGIS 4.0.3 : `addMapLayer` déclenche `autoSelectAddedLayer` → `identifyMapTool` → access violation quand une couche raster est ajoutée. Deux contournements obligatoires :

1. Passer sur l'outil **Pan** avant d'ajouter la couche
2. Utiliser `addMapLayer(layer, False)` + `root.insertLayer()` (sans auto-sélection)

Format URI : le provider WMTS natif de QGIS (`crs=...&layers=...&url=...`) échoue sur la Géoplateforme (pas de capabilities). Solution : encoder l'URL WMTS en **XYZ tiles** avec `{z}/{y}/{x}` .

```
from qgis.core import QgsProject, QgsRasterLayer
from qgis.gui import QgsMapToolPan
from qgis.utils import iface
import urllib.parse

# 1. Passer sur Pan AVANT d'ajouter la couche raster (évite le crash identifyMapTool)
pan_tool = QgsMapToolPan(iface.mapCanvas())
iface.mapCanvas().setMapTool(pan_tool)

# 2. URI XYZ (le provider WMTS natif échoue sur data.geopf.fr)
base_url = (
    "https://data.geopf.fr/wmts?SERVICE=WMTS&REQUEST=GetTile"
    "&VERSION=1.0.0&LAYER=HR.ORTHOIMAGERY.ORTHOPHOTOS"
    "&STYLE=normal&FORMAT=image/jpeg"
    "&TILEMATRIXSET=PM&TILEMATRIX={z}&TILEROW={y}&TILECOL={x}"
)
encoded = urllib.parse.quote(base_url, safe='')
```

```
uri = f"type=xyz&url={encoded}&zmin=0&zmax=19"

layer_ortho = QgsRasterLayer(uri, "Orthophoto IGN", "wms")
# 3. addToLegend=False + insertLayer manuel (évite autoSelectAddedLayer)
QgsProject.instance().addMapLayer(layer_ortho, False)
root = QgsProject.instance().layerTreeRoot()
root.insertLayer(-1, layer_ortho)  # en bas de la pile
```

2. Créer la vue PostGIS et charger les IRIS

```
import psycopg2
conn = psycopg2.connect(host='localhost', port=5433, dbname='alteris_geo',
                        user='alteris_admin', password='Alteris2026')
cur = conn.cursor()
cur.execute("""
    CREATE OR REPLACE VIEW insee_raw.v_iris_mandelieu_pop AS
    SELECT c.fid, c.geom, c.nom_iris, c.code_iris,
           ROUND(p."P22_POP "::numeric, 0)::integer AS population
    FROM insee_raw.contours_iris_2025 c
    LEFT JOIN insee_raw.iris_2025_population p ON p."IRIS " = c.code_iris
    WHERE c.code_insee = '06079'
""")
conn.commit(); conn.close()

from qgis.core import QgsVectorLayer
uri = (
    'host=localhost port=5433 dbname=alteris_geo '
    'user=alteris_admin password=Alteris2026 sslmode=disable '
    'table="insee_raw"."v_iris_mandelieu_pop" (geom) key=fid'
)
layer = QgsVectorLayer(uri, "IRIS Mandelieu-la-Napoule", "postgres")
```

3. Graduation par population (semi-transparent)

```
from qgis.core import QgsGraduatedSymbolRenderer, QgsRendererRange, QgsFillSymbol

colors = ['#EFF3FF', '#BDD7E7', '#6BAED6', '#2171B5', '#084594']
pmin, pmax = 24, 3188
step = (pmax - pmin) / 5
bounds = [pmin + i * step for i in range(6)]

ranges_def = []
for i in range(5):
    lo, hi = bounds[i], bounds[i+1]
    sym = QgsFillSymbol.createSimple({
        'color': colors[i], 'outline_color': '#555555', 'outline_width': '0.3'
    })
    sym.setOpacity(0.6)  # semi-transparent pour laisser voir l'ortho
    ranges_def.append(QgsRendererRange(lo, hi, sym,
        f"{int(lo):,}–{int(hi):,} hab.".replace(',', ' ')))

layer.setRenderer(QgsGraduatedSymbolRenderer('population', ranges_def))
```

4. Étiquettes + ajout dans le bon ordre

```
from qgis.core import (QgsPalLayerSettings, QgsTextFormat,
    QgsTextBufferSettings, QgsVectorLayerSimpleLabeling)
from qgis.PyQt.QtGui import QColor, QFont

pal = QgsPalLayerSettings()
pal.fieldName = "concat(nom_iris, '\n', population, ' hab.')"
pal.isExpression = True
pal.placement = QgsPalLayerSettings.Placement.OverPoint
fmt = QgsTextFormat()
fmt.setFont(QFont("Arial", 8)); fmt.setSize(8); fmt.setColor(QColor('#1a1a1a'))
buf = QgsTextBufferSettings()
buf.setEnabled(True); buf.setSize(1); buf.setColor(QColor('white'))
fmt.setBuffer(buf); pal.setFormat(fmt)
layer.setLabelsEnabled(True)
layer.setLabeling(QgsVectorLayerSimpleLabeling(pal))

# IRIS en haut de la pile (index 0), ortho déjà en bas
```

```
QgsProject.instance().addMapLayer(layer, False)
root = QgsProject.instance().layerTreeRoot()
root.insertLayer(0, layer)
```

5. Zoomer sur Mandelieu

```
from qgis.core import QgsRectangle, QgsCoordinateReferenceSystem, QgsCoordinateTransform

src_crs = QgsCoordinateReferenceSystem("EPSG:4326")
dst_crs = QgsCoordinateReferenceSystem("EPSG:3857")
transform = QgsCoordinateTransform(src_crs, dst_crs, QgsProject.instance())
pt_min = transform.transform(6.87, 43.51)
pt_max = transform.transform(6.99, 43.60)
extent = QgsRectangle(pt_min.x(), pt_min.y(), pt_max.x(), pt_max.y())
iface.mapCanvas().setExtent(extent)
iface.mapCanvas().refresh()
```

6. Charger le cadastre (WFS vecteur)

Pièges :

- Le provider **WFS natif QGIS** est invalide sur `data.geopf.fr` pour ce typename → utiliser le driver **OGR WFS**
- L'**API Carto** (`geo.api.gouv.fr`) a timeout (appel synchrone dans le thread QGIS, 20 s) → abandonné
- Le WFS retourne les données en **EPSG:4326** mais le canevas est en **EPSG:3857** → la couche apparaît dans le panneau mais n'est pas visible sur la carte → reproj obligatoire

```
import processing
from qgis.core import (QgsVectorLayer, QgsCoordinateReferenceSystem,
                       QgsFillSymbol, QgsSingleSymbolRenderer, QgsProject)

# Driver OGR WFS avec BBOX en paramètre URL (le provider WFS natif QGIS échoue)
url = (
    "WFS:https://data.geopf.fr/wfs/ows?SERVICE=WFS&VERSION=2.0.0&REQUEST=GetFeature"
    "&TYPENAME=CADASTRALPARCELS.PARCELLAIRE_EXPRESS:parcelle"
    "&BBOX=6.87,43.51,6.99,43.60,EPSG:4326&COUNT=2000"
)
layer_wfs = QgsVectorLayer(url, "Cadastre Mandelieu", "ogr")
```

```
# → isValid() = True, 2000 features, EPSG:4326

# Reprojection en mémoire EPSG:3857 (sinon couche invisible dans le canevas)
result = processing.run("native:reprojectlayer", {
    'INPUT': layer_wfs,
    'TARGET_CRS': QgsCoordinateReferenceSystem('EPSG:3857'),
    'OUTPUT': 'memory:'
})
layer_cad = result['OUTPUT']
layer_cad.setName("Cadastre Mandelieu")
# → 26 439 features, EPSG:3857

# Symbologie : contour orange, remplissage quasi-transparent
sym = QgsFillSymbol.createSimple({
    'color': '204,68,0,30',      # orange très transparent (alpha 30/255)
    'outline_color': '#CC4400',
    'outline_width': '0.5'
})
layer_cad.setRenderer(QgsSingleSymbolRenderer(sym))

# Ajouter au-dessus des IRIS (index 0)
QgsProject.instance().addMapLayer(layer_cad, False)
root = QgsProject.instance().layerTreeRoot()
root.insertLayer(0, layer_cad)
```

Ordre final des couches (haut → bas) :

1. Cadastre Mandelieu (vecteur WFS, EPSG:3857)
2. IRIS Mandelieu-la-Napoule (PostGIS, gradué population)
3. Orthophoto IGN (XYZ, fond)

Note cadastre : le BBOX rectangulaire déborde sur les communes voisines (Cannes à l'est, Pégomas au nord). Comportement attendu — pour restreindre au territoire communal strict, il faudrait une intersection post-chargement (pas de filtre `code_commune` disponible côté WFS Géoplateforme).

Points clés et pièges

Sujet	Constat

Bug QGIS 4.0.3 — raster + identifyMapTool	`addMapLayer` sur une couche raster → access violation si l'outil Identifier est actif. Fix : passer sur Pan + `addMapLayer(layer, False)` + `insertLayer()`
Ne pas cliquer sur la couche raster dans le panneau	Même après ajout réussi, cliquer sur la couche raster dans le panneau des couches déclenche `onActiveLayerChanged` → crash. Rester sur l'outil Pan.
Provider WMTS natif QGIS invalide	`crs=...&layers=...&url=https://data.geopf.fr/wmts` → couche invalide. Fix : encoder en XYZ avec `type=xyz&url=...{z}/{y}/{x}`
Canvas au zoom mondial au démarrage	Les tuiles ne se chargent pas. Toujours zoomer sur la zone cible après ajout.
Provider WFS natif QGIS invalide	`QgsVectorLayer(url, name, "WFS")` → couche invalide pour `CADASTRALPARCELS.PARCELLAIRE_EXPRESS:parcelle`. Fix : driver OGR WFS `QgsVectorLayer("WFS:https://...", name, "ogr")`
API Carto timeout	`geo.api.gouv.fr` : appel HTTP synchrone dans le thread QGIS → 20 s de blocage puis timeout. Driver OGR WFS Géoplateforme préférable.
CRS mismatch WFS	WFS retourne EPSG:4326, canevas EPSG:3857 → couche présente dans le panneau mais invisible sur la carte. Fix : `processing.run("native:reprojectlayer", ...)` vers EPSG:3857 en mémoire.
Colonnes INSEE avec espace	`"IRIS "`, `"COM "`, `"P22_POP "` — espace traînant partout dans `iris_2025_population`
Types text	Toutes les valeurs numériques INSEE sont en `text` — caster avec `::numeric`
Sous-requête URI QGIS	`QgsDataSourceUri.setDataSource()` sur-échappe les guillemets → couche invalide. Fix : créer une vue PostgreSQL
`sys.executable` **QGIS**	Pointe sur `qgis-bin.exe` — pip via subprocess inutilisable. Fix : `pip._internal.cli.main`
`sys.path` **paramiko**	QGIS n'inclut pas le dossier utilisateur Python. Ajouter `sys.path.insert(0, ...)` à chaque session

Généralisation à d'autres communes

```
code_insee = '06079'  # remplacer par le code cible
cur.execute(f"""
    CREATE OR REPLACE VIEW insee_raw.v_iris_pop AS
```

```
    SELECT c.fid, c.geom, c.nom_iris, c.code_iris, c.nom_commune,
           ROUND(p."P22_POP "::numeric, 0)::integer AS population
    FROM insee_raw.contours_iris_2025 c
    LEFT JOIN insee_raw.iris_2025_population p ON p."IRIS " = c.code_iris
    WHERE c.code_insee = '{code_insee}'
""")
```

Pour le cadastre, adapter le BBOX à l'emprise de la commune cible.

REX — 2026-06-21 (second run)

Résultat : 0 crash — procédure validée reproductible.

Run complet depuis un projet QGIS vide, piloté intégralement via MCP Claude Code → QGIS :

Étape	Résultat
Tunnel SSH paramiko	OK — PostGIS Alteris accessible localhost:5433
Orthophoto IGN (XYZ)	Valide — rendu fond de carte immédiat
Vue PostGIS + couche IRIS	9 features, graduation 5 classes bleu, étiquettes population
Zoom sur Mandelieu	Cadrage correct EPSG:3857
Cadastre WFS OGR + reprojection	26 439 parcelles EPSG:3857, symbologie orange

Conclusion : la procédure est stable et reproductible. Base solide pour une routine généralisable à n'importe quelle commune (`code_insee` + BBOX comme seuls paramètres variables). Potentiel : automatiser la production de fiches communales IRIS + cadastre + fond ortho à la demande depuis Claude Code.

Expertises Topologie - Claude et Qgis

Contexte PLU

Pour une couche de zones PLU topologiquement correcte, trois règles sont non-négociables :

1. **Pas de gaps** — tout le territoire communal doit être couvert, sans trou entre zones
2. **Pas d'overlaps** — une parcelle n'appartient qu'à une seule zone
3. **Géométries valides** — pas de self-intersection, pas de géométrie nulle

Ces trois vérifications + corrections sont pilotables intégralement depuis Claude Code via le MCP QGIS, sans manipulation manuelle.

Bloc 1 — Configuration du snapping avant saisie

À lancer en début de session d'édition sur une couche PLU. Configure l'environnement QGIS pour que la saisie soit topologiquement propre dès le départ.

```
from qgis.core import QgsSnappingConfig, QgsTolerance, QgsProject

proj = QgsProject.instance()
snap_cfg = proj.snappingConfig()

snap_cfg.setEnabled(True)
snap_cfg.setMode(QgsSnappingConfig.SnappingMode.AllLayers)
snap_cfg.setType(QgsSnappingConfig.SnappingType.VertexAndSegment)
snap_cfg.setTolerance(10.0)
snap_cfg.setUnits(QgsTolerance.UnitType.Pixels)
snap_cfg.setIntersectionSnapping(True)   # accrochage sur les intersections
```

```
proj.setSnappingConfig(snap_cfg)
proj.setTopologicalEditing(True)          # partage de noeuds entre couches
proj.setAvoidIntersectionsMode(
    QgsProject.AvoidIntersectionsMode.AvoidIntersectionsLayers
)  # évite automatiquement les overlaps lors de la saisie
```

Paramètres clés :

Paramètre	Valeur	Effet
`AllLayers`	mode	snap sur toutes les couches visibles
`VertexAndSegment`	type	accroche vertex ET bord
`10 px`	tolérance	adapté à la saisie courante PLU
`TopologicalEditing`	True	noeuds partagés entre polygones adjacents
`AvoidIntersectionsLayers`	mode	saisie d'un polygone qui "découpe" automatiquement ses voisins

Piège : `setType()` attend `QgsSnappingConfig.SnappingType` — pas `Qgis.SnappingType` ni `Qgis.SnappingTypes`.

Bloc 2 — Audit topologique

Routine complète : validité + overlaps + gaps sur n'importe quelle couche polygone.

```
import processing
from qgis.core import QgsProject

def audit_topo(layer, unique_id='fid', gap_threshold=0, min_overlap_area=0):
    """
    Audit topologique d'une couche polygone PLU.
    Retourne un dict avec les couches d'erreurs et un résumé.
    """
    rapport = {}

    # 1. Validité géométrique (GEOS)
    r_val = processing.run("native:checkvalidity", {
        'INPUT_LAYER': layer,   # NOTE : INPUT_LAYER, pas INPUT
```

```
    'METHOD': 2,            # GEOS
    'IGNORE_RING_SELF_INTERSECTION': False,
    'VALID_OUTPUT': 'memory:',
    'INVALID_OUTPUT': 'memory:',
    'ERROR_OUTPUT': 'memory:'
})
rapport['validite'] = {
    'valides': r_val['VALID_COUNT'],
    'invalides': r_val['INVALID_COUNT'],
    'erreurs': r_val['ERROR_COUNT'],
    'layer_invalides': r_val['INVALID_OUTPUT'],
    'layer_erreurs': r_val['ERROR_OUTPUT']
}

# 2. Overlaps
r_ov = processing.run("native:checkgeometryoverlap", {
    'INPUT': layer,
    'UNIQUE_ID': unique_id,
    'MIN_OVERLAP_AREA': min_overlap_area,
    'TOLERANCE': 8,
    'ERRORS': 'memory:',
    'OUTPUT': 'memory:'
})
rapport['overlaps'] = {
    'count': r_ov['ERRORS'].featureCount(),
    'layer_errors': r_ov['ERRORS'],
    'layer_features': r_ov['OUTPUT']
}

# 3. Gaps
r_gap = processing.run("native:checkgeometrygap", {
    'INPUT': layer,
    'UNIQUE_ID': unique_id,
    'GAP_THRESHOLD': gap_threshold,
    'TOLERANCE': 8,
    'NEIGHBORS': 'memory:',
    'ERRORS': 'memory:',
    'OUTPUT': 'memory:'
})
rapport['gaps'] = {
```

```
        'count': r_gap['OUTPUT'].featureCount(),
        'layer_gaps': r_gap['OUTPUT'],
        'layer_neighbors': r_gap['NEIGHBORS'],  # requis pour fixgeometrygap
        'layer_errors': r_gap['ERRORS']
    }

    # Résumé lisible
    print(f"=== Audit topo : {layer.name()} ({layer.featureCount()} features) ===")
    print(f"Validité  : {rapport['validite']['valides']} OK / "
          f"{rapport['validite']['invalides']} invalides")
    print(f"Overlaps  : {rapport['overlaps']['count']} erreur(s)")
    print(f"Gaps      : {rapport['gaps']['count']} trou(s)")
    ok = (rapport['validite']['invalides'] == 0 and
          rapport['overlaps']['count'] == 0 and
          rapport['gaps']['count'] == 0)
    print(f"Résultat  : {'PROPRE' if ok else 'ERREURS DETECTEES'}")

    return rapport
```

Usage :

```
from qgis.core import QgsProject
layer = QgsProject.instance().mapLayersByName("zones_plu")[0]
rapport = audit_topo(layer, unique_id='fid')
```

Piège critique : `native:checkvalidity` → paramètre `INPUT_LAYER` (pas `INPUT`). Erreur silencieuse sinon.

Bloc 3 — Correction automatique

S'appuie sur les couches d'erreurs produites par le Bloc 2.

3a. Corriger les overlaps

```
r_fix_ov = processing.run("native:fixgeometryoverlap", {
    'INPUT': layer,
    'ERRORS': rapport['overlaps']['layer_errors'],
```

```
    'UNIQUE_ID': 'fid',
    'OVERLAP_FEATURE_UNIQUE_IDX': 'gc_overlap_fid',  # champ produit par checkgeometryoverlap
    'ERROR_VALUE_ID': 'gc_error',
    'OUTPUT': 'memory:',
    'REPORT': 'memory:',
    'TOLERANCE': 8
})
layer_corrige = r_fix_ov['OUTPUT']
print(f"Overlaps corrigés : {r_fix_ov['REPORT'].featureCount()} opérations")
```

3b. Combler les gaps

3 méthodes disponibles :

Valeur	Méthode	Usage recommandé
`0`	Ajouter à la plus longue bordure en commun	PLU — zone voisine qui partage le plus de frontière
`1`	Créer une nouvelle entité	si le gap est une zone à part entière
`2`	Ajouter à la plus grande zone voisine	quand la surface prime

```
r_fix_gap = processing.run("native:fixgeometrygap", {
    'INPUT': layer,
    'NEIGHBORS': rapport['gaps']['layer_neighbors'],  # issu de checkgeometrygap
    'GAPS': rapport['gaps']['layer_gaps'],
    'METHOD': 0,    # 0 = plus longue bordure (recommandé PLU)
    'UNIQUE_ID': 'fid',
    'ERROR_ID_IDX': 'gc_errorid',
    'OUTPUT': 'memory:',
    'REPORT': 'memory:',
    'TOLERANCE': 8
})
layer_sans_gap = r_fix_gap['OUTPUT']
print(f"Gaps comblés : {r_fix_gap['REPORT'].featureCount()} opérations")
```

3c. Corriger les géométries invalides (self-intersections, anneaux)

```
r_fix_val = processing.run("native:fixgeometries", {
    'INPUT': rapport['validite']['layer_invalides'],
    'METHOD': 1,    # 1 = structure linéaire (robuste)
    'OUTPUT': 'memory:'
})
layer_valide = r_fix_val['OUTPUT']
```

Bloc 4 — Accrochage post-saisie (snapgeometries)

Quand une couche a été saisie sans snapping actif, ou importée avec de micro-décalages entre polygones adjacents. Accroche les géométries d'une couche sur une couche de référence (ex : limites communales, autre zonage).

8 comportements disponibles :

Valeur	Comportement	Usage
`0`	Aligner les nœuds, ajoute sommets si besoin	**défaut recommandé**
`1`	Point le plus proche, ajoute sommets	si nœuds mal alignés
`2`	Aligner les nœuds, sans ajout de sommet	quand on veut conserver la densité
`4`	Déplacer uniquement les extrémités (nœuds)	micro-corrections de bord
`6`	Extrémités sur extrémités uniquement	lignes/filaires

```
r_snap = processing.run("native:snapgeometries", {
    'INPUT': layer_a_corriger,
    'REFERENCE_LAYER': layer_reference,
    'TOLERANCE': 0.5,    # en unités de la couche (mètres si Lambert 93)
    'BEHAVIOR': 0,
    'OUTPUT': 'memory:'
})
layer_snapped = r_snap['OUTPUT']
```

Recommandation PLU : tolérance 0.1–0.5 m en EPSG:2154 (Lambert 93). Trop grande → déformation des géométries.

Bloc 5 — Reconstruction automatique depuis le cadastre

Quand une zone (ex : périmètre PLU, emprise de projet) a été saisie grossièrement et doit être recalée exactement sur les limites parcellaires cadastrales. Claude reconstruit le polygone par dissolution des parcelles qui chevauchent la zone rough ; l'opérateur complète ensuite manuellement les parties sans référence cadastrale (domaine public non-parcellaire : voiries, cours d'eau).

Validé le 2026-06-21 sur `06079-Minelle.gpkg` (Mandelieu-la-Napoule) : 11 parcelles intersectantes → 2 retenues (overlap ≥ 50%) → 137 004 m², 31 nœuds, géométrie valide.

Principe

1. Fixer les géométries du cadastre (WFS souvent invalide)
2. Extraire les parcelles qui intersectent la zone rough
3. Calculer le % de chevauchement pour chaque parcelle → garder celles à ≥ 50%
4. Dissoudre → multiparttosingleparts → garder la plus grande part
5. Écrire dans le GeoPackage en retirant d'abord les couches QGIS (file locking Windows)

Code complet

```
import processing
import urllib.parse
from qgis.core import (
    QgsProject, QgsVectorFileWriter, QgsWkbTypes,
    QgsFields, QgsMemoryProviderUtils
)

# Pré-requis : layer_cad (cadastre WFS reprojeté EPSG:3857)
#              layer_zone (zone rough à recaler)
#              gpkg_path (chemin du GeoPackage à écrire)
```

```
# 1. Fixer les géométries cadastrales (WFS Géoplateforme souvent invalide)
cad_fixed = processing.run("native:fixgeometries", {
    'INPUT': layer_cad,
    'METHOD': 1,    # structure linéaire, robuste
    'OUTPUT': 'memory:'
})['OUTPUT']

# 2. Extraire les parcelles qui intersectent la zone rough
parcelles_in = processing.run("native:extractbylocation", {
    'INPUT': cad_fixed,
    'PREDICATE': [0],   # intersects
    'INTERSECT': layer_zone,
    'OUTPUT': 'memory:'
})['OUTPUT']

# 3. Calculer le % de chevauchement et retenir celles ≥ 50%
geom_zone = next(layer_zone.getFeatures()).geometry()
a_inclure = []
for f in parcelles_in.getFeatures():
    inter = f.geometry().intersection(geom_zone)
    pct = inter.area() / f.geometry().area() * 100
    if pct >= 50:
        a_inclure.append(f['gid'])

print(f"{parcelles_in.featureCount()} parcelles intersectantes → "
      f"{len(a_inclure)} retenues (overlap ≥ 50%)")
print(f"GIDs retenus : {a_inclure}")

# 4. Sélectionner + dissoudre
ids_str = ', '.join(str(i) for i in a_inclure)
parcelles_sel = processing.run("native:extractbyexpression", {
    'INPUT': cad_fixed,
    'EXPRESSION': f'"gid" IN ({ids_str})',
    'OUTPUT': 'memory:'
})['OUTPUT']

dissolved = processing.run("native:dissolve", {
    'INPUT': parcelles_sel,
    'FIELD': [],
```

```
    'SEPARATE_DISJOINT': False,
    'OUTPUT': 'memory:'
})['OUTPUT']

# 5. Single part → garder la plus grande
single = processing.run("native:multiparttosingleparts", {
    'INPUT': dissolved,
    'OUTPUT': 'memory:'
})['OUTPUT']

largest = max(single.getFeatures(), key=lambda f: f.geometry().area())
print(f"Résultat : {largest.geometry().area():.0f} m², "
      f"{largest.geometry().constGet().nCoordinates()} nœuds")

layer_main = QgsMemoryProviderUtils.createMemoryLayer(
    "main", QgsFields(), QgsWkbTypes.Type.Polygon, single.crs()
)
layer_main.dataProvider().addFeature(largest)

# 6. Écrire dans le GeoPackage
# CRITIQUE : retirer toutes les couches QGIS référençant le fichier avant écriture
for name in ["06079-Minelle"]:
    for lyr in QgsProject.instance().mapLayersByName(name):
        QgsProject.instance().removeMapLayer(lyr)

options = QgsVectorFileWriter.SaveVectorOptions()
options.driverName = "GPKG"
options.layerName = "06079-Minelle"
options.actionOnExistingFile = QgsVectorFileWriter.ActionOnExistingFile.CreateOrOverwriteLayer

# writeAsVectorFormatV3 retourne un tuple de 4 valeurs
res = QgsVectorFileWriter.writeAsVectorFormatV3(
    layer_main, gpkg_path, QgsCoordinateTransformContext(), options
)
# res = (error_code, error_message, filename, layername)
if res[0] == 0:
    print(f"Écrit : {gpkg_path} → couche '{options.layerName}'")
else:
    print(f"ERREUR {res[0]} : {res[1]}")
```

```
# 7. Recharger dans QGIS pour vérification
layer_reload = QgsVectorLayer(
    f"{gpkg_path}|layername={options.layerName}",
    options.layerName, "ogr"
)
QgsProject.instance().addMapLayer(layer_reload, False)
root = QgsProject.instance().layerTreeRoot()
root.insertLayer(0, layer_reload)
iface.mapCanvas().setExtent(layer_reload.extent())
iface.mapCanvas().refresh()
print("Couche rechargée et zoomée.")
```

Pièges spécifiques

Sujet	Constat
`writeAsVectorFormatV3` **retourne 4 valeurs**	`(error_code, error_message, filename, layername)` — pas 2. Déstructurer en conséquence.
File locking Windows	QGIS garde un handle sur le fichier GeoPackage. Retirer toutes les couches référençant le fichier avec `removeMapLayer()` avant d'écrire.
`CreateOrOverwriteLayer` **pas** `CreateOrOverwriteFile`	`CreateOrOverwriteFile` supprime tout le fichier (détruisant les autres couches). `CreateOrOverwriteLayer` ne touche qu'à la couche cible.
`native:savefeatures` **échoue si fichier existe**	"A file system object already exists" — utiliser `writeAsVectorFormatV3` à la place.
MultiPolygon → `asPolygon()` **fail**	Si les parcelles retenues ne sont pas adjacentes, le dissolve produit un MultiPolygon. Appliquer `multiparttosingleparts` + `max(..., key=lambda f: f.geometry().area())` pour garder la plus grande part.
Cadastre WFS invalide	La Géoplateforme retourne parfois des géométries invalides (ex : feature 15026). Toujours fixer avec `native:fixgeometries METHOD=1` avant usage comme couche de référence.

Workflow collaboratif

Claude reconstruit le périmètre sur les limites parcellaires exactes. L'opérateur complète manuellement les parties manquantes non-parcellaires (domaine public : voiries, cours d'eau, espaces naturels non cadastrés). Les deux résultats sont complémentaires : la précision automatique sur le cadastre + le jugement humain sur le hors-cadastre.

Routine principale — Audit + Rapport

Fonction complète prête à l'emploi. Appeler par Claude en une seule commande.

```
import processing
from qgis.core import QgsProject, QgsSnappingConfig, QgsTolerance

def configurer_snapping_plu():
    proj = QgsProject.instance()
    snap_cfg = proj.snappingConfig()
    snap_cfg.setEnabled(True)
    snap_cfg.setMode(QgsSnappingConfig.SnappingMode.AllLayers)
    snap_cfg.setType(QgsSnappingConfig.SnappingType.VertexAndSegment)
    snap_cfg.setTolerance(10.0)
    snap_cfg.setUnits(QgsTolerance.UnitType.Pixels)
    snap_cfg.setIntersectionSnapping(True)
    proj.setSnappingConfig(snap_cfg)
    proj.setTopologicalEditing(True)
    proj.setAvoidIntersectionsMode(
        QgsProject.AvoidIntersectionsMode.AvoidIntersectionsLayers)
    print("Snapping PLU configuré")

def audit_topo(layer, unique_id='fid', gap_threshold=0, min_overlap_area=0):
    r_val = processing.run("native:checkvalidity", {
        'INPUT_LAYER': layer, 'METHOD': 2,
        'IGNORE_RING_SELF_INTERSECTION': False,
        'VALID_OUTPUT': 'memory:', 'INVALID_OUTPUT': 'memory:', 'ERROR_OUTPUT': 'memory:'
    })
    r_ov = processing.run("native:checkgeometryoverlap", {
        'INPUT': layer, 'UNIQUE_ID': unique_id,
        'MIN_OVERLAP_AREA': min_overlap_area, 'TOLERANCE': 8,
        'ERRORS': 'memory:', 'OUTPUT': 'memory:'
    })
    r_gap = processing.run("native:checkgeometrygap", {
        'INPUT': layer, 'UNIQUE_ID': unique_id,
```

```
        'GAP_THRESHOLD': gap_threshold, 'TOLERANCE': 8,
        'NEIGHBORS': 'memory:', 'ERRORS': 'memory:', 'OUTPUT': 'memory:'
    })
    rapport = {
        'layer': layer,
        'unique_id': unique_id,
        'validite': {
            'valides': r_val['VALID_COUNT'],
            'invalides': r_val['INVALID_COUNT'],
            'layer_invalides': r_val['INVALID_OUTPUT']
        },
        'overlaps': {
            'count': r_ov['ERRORS'].featureCount(),
            'layer_errors': r_ov['ERRORS'],
            'layer_features': r_ov['OUTPUT']
        },
        'gaps': {
            'count': r_gap['OUTPUT'].featureCount(),
            'layer_gaps': r_gap['OUTPUT'],
            'layer_neighbors': r_gap['NEIGHBORS'],
            'layer_errors': r_gap['ERRORS']
        }
    }
    ok = (r_val['INVALID_COUNT'] == 0 and
          r_ov['ERRORS'].featureCount() == 0 and
          r_gap['OUTPUT'].featureCount() == 0)
    print(f"\n=== Audit topo : {layer.name()} ===")
    print(f"  Géométries invalides : {r_val['INVALID_COUNT']}")
    print(f"  Overlaps             : {r_ov['ERRORS'].featureCount()}")
    print(f"  Gaps                 : {r_gap['OUTPUT'].featureCount()}")
    print(f"  => {'PROPRE' if ok else 'ERREURS — voir rapport'}")
    return rapport

# Usage type
# layer = QgsProject.instance().mapLayersByName("zones_plu")[0]
# configurer_snapping_plu()
# rapport = audit_topo(layer)
```

Pièges à retenir

Sujet	Constat
`INPUT_LAYER` **vs** `INPUT`	`native:checkvalidity` exige `INPUT_LAYER`, tous les autres utilisent `INPUT`. Erreur silencieuse sinon.
`QgsSnappingConfig.SnappingType`	`setType()` attend ce type précis — pas `Qgis.SnappingType` ni `Qgis.SnappingTypes`.
NEIGHBORS obligatoire pour fixgeometrygap	La couche `NEIGHBORS` produite par `checkgeometrygap` doit être conservée et passée à `fixgeometrygap`.
Tolérance snapgeometries en unités couche	Si la couche est en Lambert 93 (mètres), 0.5 = 50 cm. Ne pas confondre avec la tolérance en pixels du snapping canvas.
METHOD 0 pour PLU gaps	"Plus longue bordure en commun" est la méthode la plus cohérente pour les zonages PLU (le gap va à la zone avec laquelle il partage le plus de frontière).
AvoidIntersectionsLayers	Mode le plus adapté PLU — évite les overlaps lors de la saisie mais uniquement sur les couches cochées dans les paramètres du projet.
`writeAsVectorFormatV3` **retourne 4 valeurs**	`(error_code, error_message, filename, layername)` — pas 2.
File locking GeoPackage Windows	Retirer toutes les couches QGIS référençant le fichier avant écriture, utiliser `CreateOrOverwriteLayer`.

Statut

- **Validé le 2026-06-21** : Blocs 1–5 testés sur QGIS 4.0.3 via MCP Claude Code.
  - Bloc 1 : snapping PLU
  - Bloc 2 : `checkvalidity`, `checkgeometryoverlap`, `checkgeometrygap`
  - Bloc 5 : reconstruction `06079-Minelle.gpkg` depuis cadastre WFS → 137 004 m², 31 nœuds, valide
- **À tester** : `fixgeometryoverlap`, `fixgeometrygap`, `snapgeometries` sur une vraie couche PLU avec erreurs.