LLMOS BRIDGE

LLM Operating System Bridge

CAHIER DES CHARGES TECHNIQUE — VERSION 2.0

Spécification complète & finale pour la réalisation d'un moteur d'exécution universel permettant à tout LLM de voir, comprendre et contrôler un PC en temps réel

v2.0 — Edition Complète avec Perception Loop, IoT, Multi-Agent & Long-Term Memory

TABLE DES MATIÈRES

1. VISION & PHILOSOPHIE

1.1 Déclaration de Vision

LLMOS Bridge est le chaînon manquant entre l'intelligence des LLMs et le monde réel. C'est un protocole ouvert, un service local, et un écosystème de modules qui permettent à tout LLM — Claude, GPT-4, Mistral, Llama, ou tout modèle futur — de voir, comprendre et agir sur un PC avec la même fluidité qu'un opérateur humain.

La vision dépasse le simple moteur d'exécution. LLMOS Bridge vise à devenir l'infrastructure standard pour tout ce qu'un LLM peut faire dans le monde physique et numérique : fichiers, applications Office, navigateur, interfaces graphiques, objets connectés, capteurs IoT, bases de données, APIs — le tout via un protocole unique, sécurisé et extensible à l'infini.

> *LLMOS Bridge est à l'IA locale ce que Docker est aux containers, ce que LSP est aux éditeurs de code : une couche d'abstraction universelle qui masque la complexité et libère la puissance créatrice.*

1.2 Les 7 Principes Fondateurs

1. LLM-Agnostique

Le Bridge ne sait pas qui l'appelle. Aucune dépendance vers OpenAI, Anthropic, Google ou autre. Il communique via JSON pur. N'importe quel LLM capable de générer du JSON peut l'utiliser immédiatement.

2. Perception Native

Un LLM qui agit sans voir est aveugle. LLMOS Bridge donne au LLM des yeux via une Perception Loop continue : screenshots, OCR, état applicatif, événements système. Le LLM voit exactement ce qu'un humain voit et ajuste ses actions en conséquence.

3. Sécurité par Contrat

La sécurité n'est pas un filtre en dernier recours — c'est un contrat configuré par l'utilisateur. Le LLM opère dans un périmètre explicitement défini qu'il ne peut jamais dépasser, peu importe ce qu'il génère.

4. Stateful & Mémoriel

Le LLM n'a pas de mémoire entre les appels. Le Bridge en a. Un State Store persistant maintient l'état de chaque session, et un Long-Term Memory System permet au LLM de se souvenir des préférences, habitudes et contextes passés sur le long terme.

5. Réactif & Proactif

LLMOS Bridge ne se contente pas d'exécuter des plans — il peut surveiller le PC en temps réel et déclencher des actions autonomes quand des conditions sont remplies. Un capteur dépasse un seuil, un fichier apparaît, un email arrive — le Bridge peut réagir sans intervention humaine.

6. Extensible à l'Infini

Chaque module est un plugin indépendant. PC, IoT, cloud, hardware industriel — si quelque chose peut être contrôlé par du code Python, ça peut devenir un module LLMOS. La communauté construit l'écosystème.

7. Open Source & Souverain

Tout tourne localement. Aucune donnée ne quitte le PC de l'utilisateur. Code source ouvert, contributions bienvenues, dépendance zéro vers des services cloud propriétaires.

2. CONTEXTE & PROBLÈME À RÉSOUDRE

2.1 Le Problème Fondamental

Les LLMs de 2024 raisonnent mieux que jamais. Mais ils restent prisonniers d'une interface texte unidirectionnelle : ils génèrent, mais ne voient pas. Ils planifient, mais ne perçoivent pas le résultat de leurs actions. Ils sont intelligents mais aveugles.

Connecter un LLM à un PC aujourd'hui signifie : construire de l'infrastructure from scratch pour chaque projet, gérer manuellement la sécurité, recréer la gestion d'état, et travailler en aveugle sans retour visuel. Le résultat est toujours le même : des agents fragiles, non réutilisables, et difficiles à déboguer.

> *Un LLM sans perception visuelle est comme un chirurgien qui opère les yeux fermés. Il peut avoir le meilleur plan du monde — sans voir ce qui se passe, chaque action est un pari.*

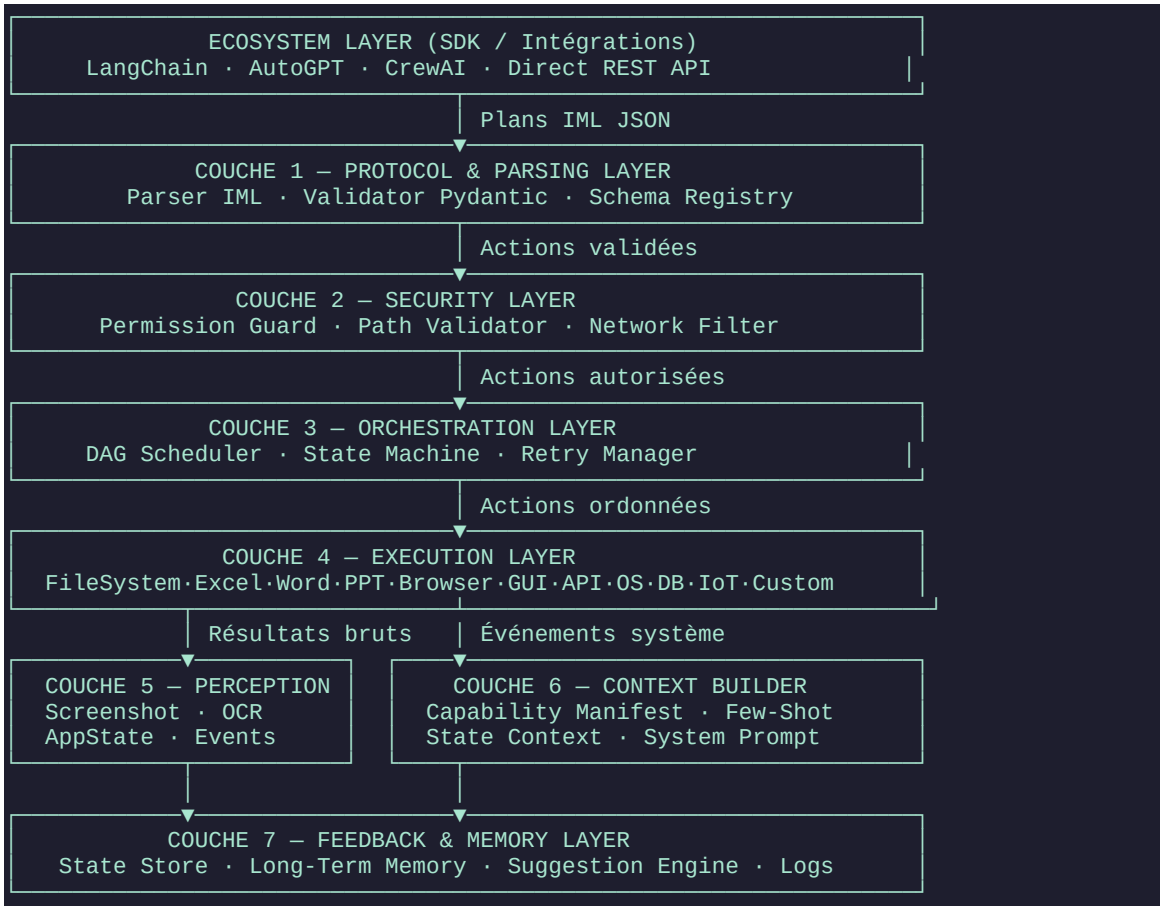
2.2 Ce que LLMOS Bridge change

Avant LLMOS Bridge	Avec LLMOS Bridge
Chaque projet recrée l'infrastructure	Une seule installation, tous les projets en bénéficient
LLM opère en aveugle	LLM voit l'écran en temps réel via Perception Loop
Pas de mémoire entre sessions	Long-Term Memory System persistant
Sécurité ad-hoc et fragile	Permission System configurable et auditable
Impossible de réagir aux événements PC	Mode réactif Event-Driven natif
Limité au monde numérique	Modules IoT pour le monde physique
Un agent = un LLM	Multi-Agent : plusieurs LLMs collaborent

3. ARCHITECTURE GLOBALE — LES 7 COUCHES

3.1 Vue d'Ensemble

L'architecture LLMOS Bridge v2 s'organise en 7 couches indépendantes. Chaque couche a une responsabilité unique et communique avec les couches adjacentes via des interfaces bien définies. L'ajout ou le remplacement d'une couche ne nécessite aucune modification des autres.



4. LE PROTOCOLE IML v2 — INSTRUCTION MARKUP LANGUAGE

4.1 Philosophie du Protocole

IML est le langage universel de LLMOS Bridge. Tout ce qu'un LLM veut faire sur un PC passe par ce protocole. Sa conception doit être simple à générer pour un LLM, riche en expressivité pour les développeurs, et stable dans le temps pour ne jamais casser l'écosystème.

> *IML v2 ajoute trois concepts fondamentaux absents de v1 : les triggers réactifs (pour le mode événementiel), les perception hints (pour guider la Perception Loop), et les memory annotations (pour alimenter le Long-Term Memory).*

4.2 Structure Complète d'une Action IML v2

```
{
  // ── Identification ──────────────────────────────────────
  "id": "a3f9",
  "label": "Écriture résultat dans Excel",
  "tags": ["excel", "finance", "q1"],

  // ── Exécution ───────────────────────────────────────────
  "action": "write_cell",
  "module": "Excel",
  "params": {
    "file": "C:/Reports/Q1.xlsx",
    "sheet": "Summary",
    "cell": "B4",
    "value": "{{result.a3f8.output}}"
  },

  // ── Orchestration ───────────────────────────────────────
  "depends_on": ["a3f8"],
  "on_error": "retry",
  "timeout": 5000,
  "retry": { "max": 3, "delay": 1000, "backoff": "exponential" },

  // ── Sécurité ────────────────────────────────────────────
  "requires_approval": false,
  "rollback": {
    "action": "write_cell",
    "params": { "cell": "B4", "value": "{{snapshot.a3f9.before}}" }
  },

  // ── Perception (NOUVEAU v2) ─────────────────────────────
  "perception": {
    "capture_before": true,
    "capture_after": true,
    "wait_for_visual_stable": true,
    "visual_assertion": "La cellule B4 doit afficher une valeur numérique"
  },

  // ── Memory (NOUVEAU v2) ─────────────────────────────────
  "memory": {
    "save_to_memory": true,
    "memory_key": "excel_q1_last_write",
    "memory_description": "Dernière valeur écrite dans Q1.xlsx cellule B4"
  }
}
```

4.3 Les Triggers Réactifs — Mode Événementiel

IML v2 introduit les triggers : des conditions qui déclenchent automatiquement un plan sans intervention du LLM. C'est le fondement du mode réactif.

```
{
  "plan_id": "surveillance_temperature",
  "mode": "reactive",
  "trigger": {
    "type": "sensor_threshold",
    "module": "RaspberryPi",
    "params": { "pin": 4, "sensor": "temperature" },
    "condition": "value > 80",
    "debounce_ms": 5000
  },
  "actions": [
    { "id": "a1", "action": "send_alert", "module": "API",
      "params": { "message": "Température critique: {{trigger.value}}°C" } },
    { "id": "a2", "action": "set_gpio", "module": "RaspberryPi",
      "depends_on": ["a1"],
      "params": { "pin": 17, "state": "HIGH", "label": "Ventilateur ON" } }
  ]
}
```

5-8. COUCHES CORE — PROTOCOL, SECURITY, ORCHESTRATION, EXECUTION

5. Protocol & Parsing Layer

Responsable de la désérialisation JSON, validation Pydantic, détection des cycles DAG, résolution automatique des modules, résolution des templates {{...}}, auto-repair des erreurs mineures, et enrichissement avec les valeurs par défaut. Aucun plan ne passe en Couche 2 sans avoir été validé et enrichi par cette couche.

6. Security Layer

Système de permissions à trois niveaux : Safe (automatique), Supervised (notification non-bloquante), Critical (validation explicite). Chaque action est comparée au Permission Profile de l'utilisateur avant exécution. Les chemins filesystem sont validés contre une whitelist. Les domaines réseau sont filtrés. Aucune action bloquée ne peut être contournée, quelle que soit la source du plan IML.

7. Orchestration Layer

DAG Scheduler basé sur NetworkX : construction d'un graphe orienté depuis les champs depends_on, exécution par niveaux topologiques avec parallélisation maximale. State Machine à 7 états par action : PENDING → WAITING_DEPS → WAITING_APPROVAL → RUNNING → SUCCESS / FAILED → RETRY / SKIPPED / ABORTED. Gestion des erreurs configurable par action via on_error.

8. Execution Layer

Dispatch des actions vers les modules appropriés. Chaque module hérite de BaseModule et implémente execute(), rollback(), health_check() et describe(). Le Module Loader découvre automatiquement les plugins installés. Voir Section 15 pour le catalogue complet des modules.

9. COUCHE 5 — PERCEPTION LOOP

9.1 Concept & Philosophie

La Perception Loop est le système qui donne au LLM ses yeux et ses sens. Sans elle, le LLM opère en aveugle — il génère des plans mais ne sait pas ce qui se passe réellement sur le PC après chaque action. Avec elle, le LLM devient un opérateur conscient qui voit, analyse et s'adapte en temps réel, exactement comme un humain devant son écran.

> *La Perception Loop transforme un LLM de 'générateur de commandes aveugle' en 'opérateur visuel intelligent'. C'est le composant qui distingue LLMOS Bridge de toutes les solutions existantes.*

9.2 Les 4 Canaux de Perception

Canal 1 — Vision Écran (ScreenPerception)

```
class ScreenPerception:
    async def capture(self, region=None) -> ScreenSnapshot:
        return ScreenSnapshot(
            screenshot_b64  = await self._capture(region),
            active_window   = self._get_active_window(),
            visible_text    = await self._ocr(),        # Tesseract
            ui_elements     = await self._detect_ui(),  # boutons, champs
            resolution      = self._get_resolution(),
            cursor_pos      = self._get_cursor(),
            timestamp       = datetime.now()
        )
```

Le LLM reçoit le screenshot en base64 et le lit via sa capacité vision (GPT-4o, Claude Vision). Il voit littéralement l'écran et peut identifier des popups d'erreur, des boutons, des messages — et adapter son plan en conséquence.

Canal 2 — Perception Applicative (AppPerception)

```
class AppPerception:
    async def capture(self) -> AppSnapshot:
        return AppSnapshot(
            active_app      = self._get_active_app(),
            open_files      = self._get_open_files(),
            selected_text   = self._get_selection(),
            clipboard       = self._get_clipboard(),
            excel_context   = self._get_excel_state(),
            # → feuille active, cellule sélectionnée, valeur
            browser_context = self._get_browser_state(),
            # → URL, titre, formulaires présents, scroll
            word_context    = self._get_word_state(),
            # → paragraphe actif, style, position curseur
        )
```

Canal 3 — Retour d'Action Visuel (ActionFeedback)

Chaque action génère automatiquement une comparaison visuelle avant/après. Le Bridge détecte les changements, les décrit textuellement, et transmet le tout au LLM.

```
class ActionFeedback:
    before_screenshot : str   # État AVANT l'action
    after_screenshot  : str   # État APRÈS l'action
    diff_regions      : list  # Zones de l'écran qui ont changé
    visual_delta      : str   # Description textuelle du changement
    assertion_result  : bool  # Assertion visuelle vérifiée ?
    # Exemple: clic sur 'Télécharger'
    # before: bouton 'Télécharger' visible
    # after:  barre de progression à 23%
    # delta:  'Le téléchargement a démarré avec succès'
```

Canal 4 — Flux Événementiel Système (SystemEventStream)

```
# Événements que le LLM peut surveiller en temps réel
class SystemEvent(Enum):
    FILE_CREATED      = 'file_created'
    FILE_MODIFIED     = 'file_modified'
    WINDOW_OPENED     = 'window_opened'
    WINDOW_CLOSED     = 'window_closed'
    DOWNLOAD_COMPLETE = 'download_complete'
    ERROR_POPUP       = 'error_popup'
    NETWORK_REQUEST   = 'network_request'
    PROCESS_STARTED   = 'process_started'
    SENSOR_THRESHOLD  = 'sensor_threshold'   # IoT
    CLIPBOARD_CHANGED = 'clipboard_changed'
    NOTIFICATION      = 'notification_appeared'
```

9.3 La Boucle Complète — Perception → Action → Perception

```
class PerceptionLoop:
    async def execute_with_perception(self, plan: IMLPlan) -> PlanResult:
        for action in plan.actions:

            # 1. Snapshot AVANT
            before = await self.perceive()

            # 2. Exécution
            result = await self.executor.run(action)

            # 3. Attente stabilisation visuelle (écran stable)
            await self._wait_visual_stable(timeout_ms=2000)

            # 4. Snapshot APRÈS
            after = await self.perceive()

            # 5. Construction feedback visuel
            feedback = self._compute_visual_feedback(before, after, result)

            # 6. Vérification assertion visuelle (si définie dans IML)
            if action.perception.visual_assertion:
                ok = await self.llm.verify_assertion(
                    action.perception.visual_assertion,
                    after.screenshot
                )
                if not ok:
                    # LLM voit que l'assertion a échoué → correction auto
                    correction = await self.llm.generate_correction(feedback)
                    await self.execute_with_perception(correction)
```

9.4 Cas d'Usage — Auto-Correction Visuelle

Scénario : le LLM clique sur 'Sauvegarder'. Le Bridge exécute le clic, résultat technique = success. Mais le screenshot révèle une popup d'erreur 'Espace disque insuffisant'. Sans Perception Loop, le plan continue. Avec Perception Loop, le LLM voit la popup, génère un plan correctif, et notifie l'utilisateur — tout en moins de 2 secondes.

> *Sans Perception Loop, les erreurs visuelles passent inaperçues. Le LLM croit avoir réussi alors que l'écran montre clairement un échec. C'est la source n°1 de bugs dans les agents LLM actuels.*

10. COUCHE 6 — CONTEXT BUILDER

10.1 Le Problème du Contexte

Un LLM sans contexte précis hallucine des actions inexistantes, invente des paramètres, génère du JSON invalide. Le Context Builder est le composant qui résout ce problème en donnant au LLM exactement ce dont il a besoin pour générer des plans IML parfaits — ni plus, ni moins.

> *Un bon contexte vaut mieux qu'un bon modèle. Un GPT-3.5 avec un contexte parfait bat un GPT-4 qui hallucine parce qu'il ne connaît pas le système.*

10.2 Les 5 Composants du Contexte

1. Capability Manifest Dynamique

Liste exacte de ce qui est disponible au moment T — uniquement les modules activés, uniquement les actions autorisées pour ce profil de permissions. Généré à chaque session. Le LLM ne peut pas inventer ce qui n'existe pas s'il voit exactement ce qui existe.

2. Règles IML Strictes

Instructions de génération non ambiguës : champs obligatoires, format des IDs, utilisation du templating, comportement en cas d'erreur. Court, précis, mémorable pour le LLM.

3. State Context Temps Réel

Ce qui se passe sur le PC maintenant : fichiers ouverts, navigateur actif, résultats des dernières actions, erreurs récentes, suggestions de correction. Le LLM sait où il en est.

4. Few-Shot Examples Adaptatifs

Exemples de plans IML valides, sélectionnés dynamiquement selon le contexte courant. Si Excel est ouvert, exemples Excel. Si le navigateur est actif, exemples Browser. Les exemples sont la technique la plus efficace pour guider la génération.

5. Long-Term Memory Résumée

Préférences de l'utilisateur, habitudes détectées, erreurs passées à éviter, formats préférés. Le LLM se souvient des sessions précédentes et s'adapte au profil de l'utilisateur.

10.3 Implémentation du Context Builder

```
class ContextBuilder:
    def build(self, session: Session) -> BuiltContext:
        return BuiltContext(
            system_prompt = self._assemble_prompt(session),
            tools         = self._generate_tools(session),
            examples      = self._select_examples(session),
        )

    def _assemble_prompt(self, session: Session) -> str:
        sections = [
            self._build_identity(),
            self._build_capabilities(session),    # Modules dispo
            self._build_iml_rules(),               # Règles de génération
            self._build_system_state(session),     # État PC actuel
            self._build_restrictions(session),     # Ce qui est interdit
            self._build_memory_summary(session),   # Mémoire long-terme
            self._build_examples(session),         # Few-shot adaptatifs
        ]
        return '  '.join(filter(None, sections))
```

10.4 Exemple de System Prompt Généré

```
Tu es un agent LLMOS Bridge v2. Tu contrôles un PC en temps réel.
Tu peux VOIR l'écran (screenshots fournis) et AGIR via des plans JSON (IML).

── MODULES DISPONIBLES ──────────────────────────────────────
[FileSystem]
  read_file(path, encoding?)          → Lit un fichier texte
  write_file(path, content, mode?)    → Écrit dans un fichier ⚠️
  create_folder(path)                 → Crée un dossier ⚠️
  search_files(directory, pattern)    → Recherche des fichiers

[Excel]
  read_range(file, sheet, range)      → Lit une plage de cellules
  write_cell(file, sheet, cell, val)  → Écrit une cellule ⚠️

⚠️ = notification utilisateur  🔴 = approbation obligatoire

── ÉTAT ACTUEL ──────────────────────────────────────────────
Fichier ouvert : C:/Finance/Q1.xlsx
Dernière action : read_range → SUCCESS (données en mémoire)
Erreur récente : Aucune

── MÉMOIRE ──────────────────────────────────────────────────
Préférence: l'utilisateur préfère les exports en PDF plutôt que DOCX
Habitude: les rapports sont toujours envoyés à direction@company.com

── RÈGLES IML ───────────────────────────────────────────────
1. IDs : 'a1', 'a2', 'a3'... (courts, séquentiels)
2. Dépendances : depends_on si tu utilises {{result.<id>.output}}
3. Ne jamais inventer une action absente de la liste ci-dessus
4. Toujours inclure 'label' pour chaque action
```

11. COUCHE 7 — FEEDBACK & LONG-TERM MEMORY LAYER

11.1 Le Feedback Enrichi

Le Feedback Layer transforme les résultats bruts de l'exécution en un paquet riche que le LLM peut analyser pour décider de sa prochaine action. En v2, ce paquet inclut les données visuelles de la Perception Loop et les suggestions de correction automatiques.

```
{
  "plan_id": "plan_001",
  "execution_summary": { "total":3, "success":2, "failed":1 },
  "results": [
    { "id":"a1", "status":"success", "output":"42500", "duration_ms":120 },
    { "id":"a2", "status":"failed",
      "error": "FileNotFoundError",
      "suggestion": "Fichier similaire trouvé: C:/Finance/budget_v2.xlsx",
      "recoverable": true }
  ],
  "visual_context": {
    "current_screenshot": "base64...",
    "visible_elements": ["Dialog: Erreur ouverture fichier", "Bouton OK"],
    "active_window": "Microsoft Excel"
  },
  "next_hint": "Une popup d erreur Excel est visible. Fermer avant de continuer."
}
```

11.2 Long-Term Memory System

C'est l'une des fonctionnalités les plus puissantes de LLMOS Bridge v2. Le LLM accumule de la connaissance sur l'utilisateur, ses habitudes, ses préférences, ses erreurs passées — et s'améliore session après session.

```
class LongTermMemory:
    # Catégories de mémoire
    PREFERENCES  = 'preferences'   # Préférences utilisateur
    HABITS       = 'habits'        # Habitudes détectées
    ERRORS       = 'errors'        # Erreurs passées à éviter
    KNOWLEDGE    = 'knowledge'     # Faits appris sur le système
    CONTACTS     = 'contacts'      # Emails, noms, rôles détectés
    SHORTCUTS    = 'shortcuts'     # Raccourcis et chemins fréquents

    async def remember(self, key: str, value: str,
                       category: str, confidence: float):
        '''Sauvegarde un fait en mémoire long-terme'''

    async def recall(self, query: str, limit=5) -> list[Memory]:
        '''Récupère les souvenirs pertinents pour une requête'''
        # Recherche sémantique sur les mémoires stockées
```

Exemples de mémoires accumulées automatiquement :

- L'utilisateur préfère les PDF aux DOCX pour les rapports
- Les fichiers Excel sont toujours dans C:/Finance/
- Les emails de rapport vont à direction@company.com
- L'action delete_file a échoué 3 fois sur des fichiers ouverts — attendre qu'ils soient fermés
- L'utilisateur travaille principalement entre 9h et 18h

11.3 Suggestion Engine

Le Suggestion Engine analyse chaque erreur et génère automatiquement des pistes de correction utilisables directement par le LLM dans sa prochaine itération.

Type d'erreur	Suggestion générée
FileNotFoundError: budget.xlsx	Fichier similaire détecté: budget_2024.xlsx dans C:/Finance/
ElementNotFound: #submit-btn	Sélecteur alternatif détecté: button[type=submit] ou texte 'Envoyer'
PermissionError: C:/Windows/	Chemin autorisé le plus proche: C:/Users/Documents/
TimeoutError: navigate(url)	URL inaccessible. Vérifier la connexion réseau ou augmenter timeout
ExcelError: feuille 'Data'	Feuilles disponibles dans ce fichier: Sheet1, Summary, Q1, Q2

16. MODULE IoT — RASPBERRY PI & EMBARQUÉ

16.1 Vision IoT

LLMOS Bridge ne se limite pas au monde numérique. Via les modules IoT, le LLM peut interagir avec le monde physique : lire des capteurs, contrôler des actionneurs, dialoguer avec des microcontrôleurs. Le même protocole IML, les mêmes règles de sécurité, le même SDK LangChain — mais pour du matériel physique.

> *Un Raspberry Pi est un PC Linux. LLMOS Bridge s'installe dessus exactement comme sur un PC Windows ou Mac. Le module IoT est simplement un plugin supplémentaire qui expose les GPIO, les capteurs I2C, UART, SPI et les protocoles sans fil.*

16.2 Architectures Supportées

Matériel	Interface	Protocole	Cas d'usage
Raspberry Pi 3/4/5	GPIO, I2C, SPI, UART	RPi.GPIO, gpiozero	Domotique, robotique, monitoring
Arduino	USB Serial	pyserial	Capteurs, actionneurs simples
ESP32 / ESP8266	WiFi, Bluetooth	MQTT, WebSocket	Objets connectés sans fil
Capteurs I2C	I2C Bus	smbus2	Température, humidité, pression
Relais & actionneurs	GPIO	gpiozero	Contrôle ON/OFF d'appareils
Caméra Pi	CSI / USB	picamera2	Vision, surveillance
Réseau de capteurs	MQTT Broker	paho-mqtt	Infrastructure IoT distribuée

16.3 Actions du Module RaspberryPi

```
ACTIONS IoT:

  read_gpio(pin)                    → Lit état HIGH/LOW d'une broche
  set_gpio(pin, state)              → Définit état d'une broche ⚠️
  read_temperature(pin, sensor_type)  → Lit température (DS18B20, DHT22)
  read_humidity(pin, sensor_type)     → Lit humidité
  read_analog(channel)              → Lit valeur analogique (ADC MCP3008)
  read_i2c(address, register)       → Lit registre I2C
  write_i2c(address, register, value) → Écrit registre I2C ⚠️
  send_serial(port, message)        → Envoie message UART
  read_serial(port, timeout)        → Lit réponse UART
  publish_mqtt(topic, payload)      → Publie message MQTT ⚠️
  subscribe_mqtt(topic)             → S'abonne à un topic MQTT
  capture_image(output_path)        → Capture image caméra
  measure_distance(trigger_pin, echo) → Mesure distance ultrason
  control_servo(pin, angle)         → Contrôle servo-moteur ⚠️
```

16.4 Exemple — Agent de Monitoring Industriel

```
{
  "plan_id": "monitoring_salle_serveur",
  "mode": "reactive",
  "trigger": {
```

```
    "type": "sensor_threshold",
    "action": "read_temperature",
    "module": "RaspberryPi",
    "params": { "pin": 4, "sensor_type": "DS18B20" },
    "condition": "value > 75",
    "poll_interval_ms": 5000
  },
  "actions": [
    { "id":"a1", "action":"set_gpio", "module":"RaspberryPi",
      "params":{"pin":17,"state":"HIGH","label":"Ventilateur urgence ON"} },
    { "id":"a2", "action":"send_email_smtp", "module":"API",
      "depends_on":["a1"],
      "params":{
        "recipient":"admin@datacenter.com",
        "subject":"ALERTE TEMPERATURE CRITIQUE",
        "body":"Température: {{trigger.value}}°C — Ventilateur activé"
      }
    },
    { "id":"a3", "action":"write_cell", "module":"Excel",
      "depends_on":["a1"],
      "params":{
        "file":"C:/Logs/temperature_log.xlsx",
        "sheet":"Alertes",
        "cell":"auto_next_row",
        "value":"{{trigger.timestamp}} | {{trigger.value}}°C | CRITIQUE"
      }
    }
  ]
}
```

17. MODE RÉACTIF — EVENT-DRIVEN ARCHITECTURE

17.1 Au-delà de l'Exécution à la Demande

Le Bridge v1 fonctionne en mode request-response : le LLM envoie un plan, le Bridge exécute, retourne le résultat. Le mode réactif inverse ce paradigme : le Bridge surveille en permanence des conditions et déclenche des plans automatiquement quand ces conditions sont remplies.

C'est la différence entre un assistant qui répond quand on lui parle et un assistant qui surveille et agit de façon autonome.

17.2 Types de Triggers Supportés

Type de Trigger	Exemple	Module
Fichier système	Un fichier .xlsx apparaît dans C:/Downloads/	FileSystem
Seuil capteur	Température dépasse 80°C	IoT/RaspberryPi
Email reçu	Email avec sujet 'URGENT' reçu	API/Email
Heure planifiée	Tous les jours à 8h00	Scheduler
Condition GUI	Une popup d'erreur apparaît sur l'écran	Perception/GUI
Requête HTTP	Webhook entrant sur /trigger/{id}	API/HTTP
Processus	Application X se ferme	OS
MQTT	Message reçu sur topic 'sensors/#'	IoT/MQTT
Intervalle	Toutes les 30 minutes	Scheduler
Condition composite	Température > 70 ET humidité > 80%	Multi-trigger

17.3 Architecture du Reactive Engine

```
class ReactiveEngine:
    def __init__(self):
        self.watchers: dict[str, TriggerWatcher] = {}
        self.plans:    dict[str, IMLPlan]        = {}

    async def register_reactive_plan(self, plan: IMLPlan):
        '''Enregistre un plan réactif et démarre son watcher'''
        watcher = TriggerWatcher.from_trigger(plan.trigger)
        watcher.on_triggered(lambda event: self._execute(plan, event))
        self.watchers[plan.plan_id] = watcher
        await watcher.start()

    async def _execute(self, plan: IMLPlan, event: TriggerEvent):
        # Injection de l'événement dans le contexte du plan
        enriched_plan = self._inject_trigger_context(plan, event)
        await self.orchestrator.execute(enriched_plan)
```

18. MULTI-AGENT COORDINATION

18.1 Vision

LLMOS Bridge supporte nativement la coordination de plusieurs agents LLM travaillant en parallèle ou en séquence. Un agent orchestrateur peut déléguer des sous-tâches à des agents spécialisés. Chaque agent opère dans son propre contexte de session mais partage le même State Store.

18.2 Patterns Multi-Agent Supportés

Pattern 1 — Orchestrateur / Sous-agents

Un agent principal reçoit une tâche complexe, la découpe en sous-tâches, et délègue chacune à un agent spécialisé. L'orchestrateur consolide les résultats.

```
// Agent orchestrateur délègue à des agents spécialisés
{ "action": "delegate",
  "params": {
    "agent": "excel_specialist",
    "task": "Extraire et analyser les données Q1",
    "context": "{{state.context.current_file}}"
  }
}
```

Pattern 2 — Pipeline Séquentiel

La sortie d'un agent devient l'entrée du suivant. Agent A analyse un document, Agent B rédige un résumé, Agent C l'envoie par email.

Pattern 3 — Parallélisation

Plusieurs agents traitent simultanément différentes parties d'une tâche. Gain de temps significatif sur les workflows complexes.

18.3 Isolation & Sécurité Multi-Agent

Chaque agent opère dans sa propre session isolée. Les agents ne peuvent pas accéder à l'état des autres sessions sauf via des données explicitement partagées. Les permissions s'appliquent individuellement à chaque agent.

19-20. SELF-HEALING, RÉSILIENCE & DASHBOARD

19. Self-Healing & Résilience

LLMOS Bridge intègre des mécanismes de résilience automatique qui réduisent considérablement le besoin d'intervention humaine face aux erreurs courantes.

Mécanisme	Déclencheur	Comportement
Auto-retry	TimeoutError, réseau instable	Retry exponentiel: 1s, 2s, 4s, 8s (max configurable)
Path healing	FileNotFoundError	Fuzzy matching filesystem → suggestion du bon chemin au LLM
Selector healing	ElementNotFound (Web/GUI)	Analyse DOM / OCR → sélecteurs alternatifs proposés
Session recovery	Crash du Bridge	Restauration automatique depuis le State Store au redémarrage
Visual assertion retry	Assertion visuelle échouée	Re-tentative de l'action après analyse du screenshot
Deadlock detection	Plan bloqué > timeout	Détection et interruption propre avec rapport d'erreur
Memory overflow	State Store > taille max	Archivage automatique des anciennes sessions

20. Dashboard de Monitoring

Une interface web légère (localhost:7339) permet à l'utilisateur de surveiller le Bridge en temps réel, de gérer les sessions, de voir les logs, et d'approuver les actions critiques.

- Vue temps réel des actions en cours d'exécution
- Historique des sessions et des plans exécutés
- Visualisation du State Store et de la Long-Term Memory
- Gestion des permissions et des profils
- Centre de notifications pour les approbations requises
- Métriques de performance : taux de succès, durées, erreurs fréquentes
- Logs en streaming avec filtrage par session, module, niveau

21-22. SÉCURITÉ AVANCÉE & LOGGING

21. Sécurité — Modèle de Menace Complet

Menace	Vecteur	Mitigation LLMOS Bridge
LLM mal aligné	Plan malveillant	Permission Profile + validation stricte avant exécution
Injection de prompt	Données web/fichiers influençant le LLM	Sanitisation des sorties de modules avant injection contexte
Path traversal	../../../etc/passwd dans params	Normalisation + validation contre whitelist
Processus local malveillant	App locale appelle le Bridge	Token auth Bearer requis sur tout endpoint
Exfiltration données	Module API vers domaine malveillant	Whitelist domaines réseau + log de toutes les requêtes HTTP
Escalade privilèges	Accès System32, registre	Blocklist chemins système + actions bloquées par profil
Replay attack	Réutilisation d'un plan intercepté	Timestamp + nonce sur chaque plan, TTL 60 secondes
Abus du Bridge	Boucle infinie de plans	Rate limiting: max 100 actions/minute par session

22. Logging & Audit Trail

Chaque action est loguée avec le niveau de détail approprié. Les actions Critical génèrent un enregistrement d'audit permanent qui ne peut pas être supprimé, même par l'administrateur.

Niveau	Contenu	Rétention
TRACE	Détails parsing, templates, résolution modules	1 semaine
DEBUG	Params complets, contexte complet	2 semaines
INFO	Actions exécutées, statuts, durées	3 mois
WARN	Retries, approbations refusées, timeouts	6 mois
ERROR	Échecs, erreurs sécurité, exceptions	1 an
AUDIT	Toutes actions Critical + approbations utilisateur	Permanent — non supprimable

23. DISTRIBUTION OPEN SOURCE & PLUGIN REGISTRY

23.1 Les Trois Packages

```
# Service Core — installé par l'utilisateur final
pip install llmos-bridge
llmos-bridge start            # Lance le daemon
llmos-bridge dashboard        # Ouvre le dashboard web

# SDK LangChain — installé par les développeurs
pip install langchain-llmos

# Modules additionnels — communauté
pip install llmos-module-excel
pip install llmos-module-raspberry
pip install llmos-module-sap
pip install llmos-module-autocad
```

23.2 Plugin Registry

Un registre central (registry.llmos-bridge.io) référence tous les modules publiés par la communauté. Les utilisateurs peuvent découvrir, évaluer, et installer des modules en une commande.

```
# Découverte de modules
llmos-bridge registry search excel
llmos-bridge registry search iot

# Installation d'un module tiers
llmos-bridge plugin install llmos-module-sap
llmos-bridge plugin install llmos-module-autocad

# Liste des modules installés
llmos-bridge plugin list
```

23.3 Critères de Qualité pour la Publication

Un module pour être référencé dans le Registry officiel doit respecter les critères suivants :

1. Implémenter BaseModule avec execute(), describe() et health_check()
2. Déclarer toutes ses actions dans ACTIONS avec descriptions LLM-friendly
3. Fournir au moins 2 examples par action dans ActionSpec
4. Coverage de tests unitaires > 70%
5. Documentation README avec installation et exemples
6. Respect du versioning sémantique
7. Aucune dépendance vers des services cloud propriétaires non optionnels

24. ROADMAP COMPLÈTE — 7 PHASES

Phase 1 — Foundation Core (Semaines 1-4)

Objectif : Bridge Service fonctionnel. Un agent LangChain peut manipuler des fichiers et recevoir un feedback structuré.

- Protocol Layer : IML Parser, Validateur Pydantic v2, résolution modules
- Security Layer basique : Permission Profile, Permission Guard
- Orchestration Layer : DAG Scheduler (NetworkX), State Machine
- Module FileSystem complet : 14 actions
- Module OS/Terminal : run_command, processes, open_application
- HTTP Server (FastAPI) + WebSocket Server
- State Store : persistance JSON sur disque
- CLI : llmos-bridge start/stop/status
- SDK LangChain v0.1 : get_tools() basique

Phase 2 — Office Suite (Semaines 5-8)

Objectif : Le LLM peut lire, modifier et créer des documents Office complets.

- Module Excel : cells, ranges, charts, formulas, export PDF
- Module Word : paragraphs, tables, styles, export PDF
- Module PowerPoint : slides, images, export PDF
- Rollback System : snapshots avant/après, pile de rollback
- Feedback Layer : Suggestion Engine, retour structuré enrichi
- SDK LangChain v0.2 : auto-génération tools depuis Capability Manifest

Phase 3 — Perception & Web (Semaines 9-13)

Objectif : Le LLM voit l'écran en temps réel et peut interagir avec le web.

- Perception Loop complète : ScreenPerception, AppPerception, ActionFeedback
- Module Browser (Playwright) : navigate, click, scrape, download
- Module GUI (PyAutoGUI + Tesseract OCR) : click_image, type_text, find_on_screen
- Module API/HTTP : http_get/post, send_email, webhooks
- Context Builder v1 : Capability Manifest, règles IML, state context
- SDK LangChain v0.3 : streaming WebSocket, LangGraph integration

Phase 4 — Memory & Réactivité (Semaines 14-17)

Objectif : Le LLM se souvient et réagit aux événements autonomement.

- Long-Term Memory System : préférences, habitudes, erreurs passées
- Context Builder v2 : few-shot adaptatifs, mémoire injectée
- Mode Réactif Event-Driven : triggers fichier, sensor, email, planifié
- Module IoT — Raspberry Pi : GPIO, I2C, UART, capteurs courants
- Module Database : SQLite, PostgreSQL, MySQL

Phase 5 — Sécurité & Stabilité (Semaines 18-21)

Objectif : Système robuste, sécurisé et prêt pour la production.

- Self-Healing : auto-retry, path healing, selector healing, session recovery
- Dashboard de Monitoring : interface web localhost:7339
- Multi-Agent Coordination : sessions isolées, patterns orchestrateur/sous-agents
- Permission System complet : profiles multiples, GUI de configuration
- Audit Trail permanent pour actions Critical
- Tests E2E : > 200 scénarios couvrant tous les modules
- Rate limiting, protection anti-abus

Phase 6 — Open Source Launch (Semaines 22-25)

Objectif : Publication open source avec une communauté active dès le premier jour.

- Publication PyPI : llmos-bridge, langchain-llmos
- Plugin Registry : registry.llmos-bridge.io
- 10 modules officiels publiés au lancement
- Connecteurs : AutoGPT Plugin, CrewAI Tool, Direct REST API
- Documentation complète : ReadTheDocs, tutoriels, exemples
- 5 agents de démonstration documentés end-to-end

Phase 7 — Ecosystem Growth (Mois 6-12)

Objectif : Faire de LLMOS Bridge le standard de facto pour les agents LLM sur desktop.

- Module SAP, AutoCAD, Photoshop (via plugins communauté)
- Support natif ARM / Apple Silicon pour les Raspberry Pi avancés
- Interface voix : commande vocale → plan IML via Whisper
- Module Cloud : AWS S3, Google Drive, OneDrive
- Certification de modules tiers : badge 'LLMOS Certified'
- Conférences, blog posts, démos vidéo pour adoption communauté

25. STACK TECHNIQUE COMPLÈTE

Composant	Technologie	Version	Justification
Langage principal	Python	3.11+	Ubiquité, écosystème IA, LLMs la connaissent parfaitement
Validation IML	Pydantic	v2	Validation JSON ultra-rapide, génération JSONSchema auto
HTTP Server	FastAPI	0.110+	Async natif, OpenAPI auto-généré, performance
WebSocket	websockets + FastAPI	—	Streaming temps réel pour Perception Loop
DAG Scheduler	NetworkX	3.x	Graphes orientés, topological_generations()
Async Runtime	asyncio + uvicorn	—	Concurrence sans overhead threading
Logging	structlog	23.x	Logs JSON structurés, context binding
State Store	SQLite + aiosqlite	—	Léger, local, sans serveur, async
Long-Term Memory	ChromaDB (vectoriel)	0.4+	Recherche sémantique sur les souvenirs
Screen Capture	mss	7.x	Screenshot multi-plateforme performant
OCR	pytesseract + Tesseract	5.x	Lecture texte depuis screenshots
UI Automation	PyAutoGUI + pygetwindow	—	Contrôle GUI multi-plateforme
Browser	Playwright	1.40+	Plus moderne et stable que Selenium
Excel	openpyxl + xlwings	—	openpyxl fichiers, xlwings Excel ouvert
Word/PPT	python-docx + python-pptx	—	Manipulation Office sans dépendance COM
IoT GPIO	RPi.GPIO + gpiozero	—	Abstraction GPIO Raspberry Pi
IoT MQTT	paho-mqtt	1.6+	Standard MQTT pour IoT
HTTP Client	httpx	0.27+	Async, HTTP/2, meilleur que requests
Dashboard UI	FastAPI + HTMX	—	Interface légère sans framework JS lourd
Tests	pytest + pytest-asyncio	—	Tests unitaires et E2E async
Packaging	Poetry + PyPI	—	Gestion dépendances moderne

26. KPIs & MÉTRIQUES DE SUCCÈS

KPI	Cible Phase 1	Cible Launch	Cible 1 an
Intégration SDK LangChain	< 15 min	< 5 min	< 2 min
Latence action simple	< 300ms	< 150ms	< 100ms
Taux de succès plan simple	90%	99%	99.5%
Taux de succès plan complexe (5+ actions)	75%	92%	97%
JSON valide généré par LLM (avec contexte)	85%	97%	99%
Modules built-in	2	10	15+
Modules tiers communauté	0	5	50+
Coverage tests	70%	85%	90%
RAM service en attente	< 100MB	< 50MB	< 40MB
Étoiles GitHub	—	500+	5000+
Stars PyPI (téléchargements/mois)	—	1000+	50000+

27. ANNEXES — EXEMPLES COMPLETS

27.1 Agent Complet — Workflow Financier avec Perception

```
// Scénario: Analyser Q1.xlsx, créer rapport Word, vérifier visuellement, envoyer
{
  "plan_id": "workflow_financier_q1",
  "description": "Rapport Q1 complet avec vérification visuelle",
  "actions": [
    {
      "id":"a1", "action":"read_range", "module":"Excel",
      "params":{"file":"C:/Finance/Q1.xlsx","sheet":"Data","range":"A1:E50"},
      "perception":{"capture_after":true},
      "label":"Lecture données Q1"
    },
    {
      "id":"a2", "action":"screenshot", "module":"GUI",
      "depends_on":["a1"],
      "params":{},
      "label":"Capture état Excel après lecture"
    },
    {
      "id":"a3", "action":"create_document", "module":"Word",
      "depends_on":["a1"],
      "params":{"output_path":"C:/Reports/rapport_Q1.docx"},
      "label":"Création document rapport"
    },
    {
      "id":"a4", "action":"append_text", "module":"Word",
      "depends_on":["a3"],
      "params":{
        "path":"C:/Reports/rapport_Q1.docx",
        "text":"Rapport Q1\nDonnées: {{result.a1.output}}",
        "style":"Normal"
      },
      "memory":{"save_to_memory":true,"memory_key":"last_q1_report_path"},
      "label":"Rédaction contenu rapport"
    },
    {
      "id":"a5", "action":"send_email_smtp", "module":"API",
      "depends_on":["a4"],
      "requires_approval":true,
      "on_error":"ask_llm",
      "params":{
        "recipient":"direction@company.com",
        "subject":"Rapport Q1 2024 — Automatisé par LLM",
        "body":"Rapport généré automatiquement. Voir PJ.",
        "attachments":["C:/Reports/rapport_Q1.docx"]
      },
      "label":"Envoi email direction"
    }
  ]
}
```

27.2 Agent IoT — Surveillance Température

```
{
  "plan_id": "surveillance_datacenter",
  "mode": "reactive",
  "trigger": {
    "type": "sensor_threshold",
    "action": "read_temperature",
    "module": "RaspberryPi",
    "params": {"pin": 4, "sensor_type": "DS18B20"},
    "condition": "value > 75",
    "poll_interval_ms": 5000
  },
```

```
  "actions": [
    {"id":"a1","action":"set_gpio","module":"RaspberryPi",
     "params":{"pin":17,"state":"HIGH"},"label":"Ventilateur ON"},
    {"id":"a2","action":"send_email_smtp","module":"API","depends_on":["a1"],
     "requires_approval":false,
     "params":{"recipient":"admin@dc.com","subject":"ALERTE TEMP",
               "body":"Temp: {{trigger.value}}°C à {{trigger.timestamp}}"}}
  ]
}
```

27.3 JSONSchema IML v2 Officiel

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "https://llmos-bridge.io/schemas/iml/v2.0.json",
  "title": "IMLPlan v2",
  "type": "object",
  "required": ["actions"],
  "properties": {
    "plan_id":          { "type": "string" },
    "protocol_version": { "type": "string", "pattern": "^\\d+\\.\\d+$" },
    "session_id":       { "type": "string" },
    "mode":             { "type": "string", "enum":
["sequential","reactive","parallel"] },
    "trigger":          { "$ref": "#/$defs/Trigger" },
    "actions": {
      "type": "array", "minItems": 1,
      "items": { "$ref": "#/$defs/IMLAction" }
    }
  },
  "$defs": {
    "IMLAction": {
      "required": ["id", "action", "params"],
      "properties": {
        "id":                 { "type": "string" },
        "action":             { "type": "string" },
        "module":             { "type": "string" },
        "params":             { "type": "object" },
        "depends_on":         { "type": "array" },
        "on_error":           { "enum": ["abort","skip","retry","ask_llm"] },
        "requires_approval":  { "type": "boolean" },
        "timeout":            { "type": "integer" },
        "retry":              { "$ref": "#/$defs/RetryPolicy" },
        "rollback":           { "type": "object" },
        "perception":         { "$ref": "#/$defs/PerceptionConfig" },
        "memory":             { "$ref": "#/$defs/MemoryConfig" },
        "label":              { "type": "string" },
        "tags":               { "type": "array" }
      }
    }
  }
}
```

CONCLUSION

LLMOS Bridge v2 est une infrastructure complète pour l'ère des agents LLM autonomes. Au-delà d'un moteur d'exécution, c'est un système qui donne au LLM toutes les capacités d'un opérateur humain : voir l'écran, agir sur les fichiers et applications, se souvenir des sessions passées, réagir aux événements en temps réel, et interagir avec le monde physique via l'IoT.

L'architecture en 7 couches indépendantes garantit qu'aucune évolution future — nouveau LLM, nouveau module, nouveau matériel — ne nécessite de refactoring du core. Le protocole IML v2 est conçu pour durer et pour que la communauté construise dessus pendant des années.

Trois choses ont été optimisées en priorité dans cette conception : la qualité de génération IML par le LLM (Context Builder + few-shot), la robustesse face aux erreurs imprévues (Self-Healing + Perception Loop), et la facilité d'extension par la communauté (BaseModule contract + Plugin Registry).

> *L'objectif final : que LLMOS Bridge devienne aussi naturel pour un développeur d'agent LLM que Docker l'est pour un développeur backend. Un standard qu'on installe, qu'on étend, et sur lequel on compte.*

— LLMOS Bridge Project — Open Source — v2.0 —