Ablation Test 1 — External MCP tools disabled (RDKit / CCDC / PubChem / Google search)

Goal

- **Ablation condition**: During this ablation run, **no calls are made** to the external MCP tools:
  - **RDKit** (via `chemistry` MCP server)
  - **CCDC database** (via `ccdc` MCP server)
  - **PubChem** (via `pubchem` MCP server)
  - **Google search** (via `enhanced_websearch` MCP server)
- **Expected behavior**: `generic_main.py` can **resume** from existing intermediate artifacts (e.g., stitched markdown, extracted hints files) and **jump directly to KG building steps** while external MCP tools are disabled.
- **Output isolation**: write ablation outputs into a **different output folder** while **reusing** existing intermediate results.

Important nuance (scope clarity)

- **Fast ablation (default plan)**: reuse previously generated intermediates (including extraction hints) and ensure that **this ablation run itself** makes **zero external MCP calls**.
- **Strict ablation (follow-up option)**: rerun the extraction steps too with external MCP tools disabled. This is more “pure” scientifically, but slower and may require prompt/tooling guardrails.

Repo facts we will rely on (already implemented behavior)

- Many pipeline steps **already skip** work when outputs/markers exist:
  - `top_entity_extraction` skips if `data/<hash>/top_entities.txt` exists.
  - `stitching` skips if `data/<hash>/<hash>_stitched.md` exists.
  - `main_ontology_extractions` skips if `data/<hash>/.main_ontology_extractions_done` exists.
  - `main_kg_building` skips if `data/<hash>/.main_kg_building_done` exists.
  - `extensions_extractions`, `extensions_kg_building`, `mop_derivation` have similar marker files.
- KG-building steps construct agents via `BaseAgent(mcp_tools=[...], mcp_set_name=...)`.
  - **Critical**: `BaseAgent` opens MCP sessions for *every name in `mcp_tools`*; therefore, to “disable” a tool we must **remove it from the `mcp_tools` list** (not only from the MCP config JSON).

Required ablation mechanics

1) “Disable external MCP tools” mechanism (implementation plan)

Add a single, centrally enforced tool-filter that applies to **all agent invocations** in pipeline steps:

- **Blocked tool names** (hard-coded for this ablation):
  - `chemistry` (RDKit)
  - `ccdc`

- `pubchem`
- `enhanced_websearch` (Google Serper)

- **Implementation approach**:
  - Add a small helper in `src/pipelines/utils/` (e.g., `mcp_filters.py`):
    - `filter_mcp_tools(requested: list[str], blocked: set[str]) -> list[str]`
    - logs before/after (so we can prove no external tools are used)
  - Each pipeline step that constructs `BaseAgent` must call this filter.

- **Files/steps that MUST apply the filter** (at minimum):
  - `src/pipelines/top_entity_kg_building/build.py`
  - `src/pipelines/main_kg_building/build.py`
  - `src/pipelines/extensions_kg_building/build.py`
  - Any "agent" used in `mop_derivation` (either directly or indirectly) that wires MCP tools (we'll locate and patch those call sites).

2) Output isolation while reusing intermediates (implementation plan)

We will avoid changing `data_dir` globally (since intermediates are under `data/<hash>/...`) and instead write new ablation outputs under a dedicated subfolder:

- **Ablation output root**:
  - `data/<hash>/ablation/Ablation_test_One_external_disabled/`

- **Principle**:
  - **Reads** come from the existing baseline locations (e.g., `data/<hash>/mcp_run/...`, `data/<hash>/mcp_run_ontomops/...`, stitched paper, etc.)
  - **Writes** go to the ablation output root (responses/prompts/TTL outputs/markers).

- **Concrete path changes to implement**:
  - For `main_kg_building`:
    - keep reading hints from: `data/<hash>/mcp_run/iter{N}_hints_*.txt`
    - write new:
      - prompts → `.../ablation/<tag>/prompts/...`
      - responses → `.../ablation/<tag>/responses/...`
      - intermediate TTLs → `.../ablation/<tag>/intermediate_ttl_files/...`
      - completion marker → `.../ablation/<tag>/.main_kg_building_done`
  - For `extensions_kg_building`:
    - keep reading extraction from: `data/<hash>/mcp_run_ontomops/` and `data/<hash>/mcp_run_ontospecies/`
    - write outputs into:
      - `.../ablation/<tag>/ontomops_output/...`
      - `.../ablation/<tag>/ontospecies_output/...`
      - marker → `.../ablation/<tag>/.extensions_kg_building_done`
  - For `mop_derivation`:

        - (Option A) write all derived artifacts under `.../ablation/<tag>/cbu_derivation/...` and marker under `.../ablation/<tag>/.mop_derivation_done`
        - (Option B, if too invasive initially) skip mop derivation for this ablation and document as "not executed" (but the stated goal says "all tasks", so Option A is preferred).

3) "Jump directly to KG building" in `generic_main.py` (implementation plan)

Add CLI switches that enable a clean ablation run without re-running earlier steps:

- **New CLI args** to add in `generic_main.py`:
    - `--start-step <step_name>`: execute steps starting from this step (inclusive).
        - For the fast ablation run, we will typically use `--start-step main_kg_building` or `--start-step extensions_kg_building`.
    - `--run-tag <string>` or `--ablation-name <string>`:
        - used to form the ablation output root folder.
    - `--disable-external-mcp`:
        - sets `blocked_mcp_tools={"chemistry","ccdc","pubchem","enhanced_websearch"}` and passes it into step configs as `blocked_mcp_tools`.
- **How it will work internally**:
    - `generic_main.run_pipeline(...)` will:
        - filter the `steps` list based on `--start-step`
        - pass `run_tag` and `blocked_mcp_tools` into every step's `step_config`

Baseline artifacts required (for "fast ablation" reuse)

Before running ablation, the following should already exist for each `doi_hash`:

- **Top entities**: `data/<hash>/mcp_run/iter1_top_entities.json`
- **Main ontology extraction hints** (depending on iterations present):
    - `data/<hash>/mcp_run/iter2_hints_*.txt`
    - `data/<hash>/mcp_run/iter3_hints_*.txt`
    - `data/<hash>/mcp_run/iter4_hints_*.txt`
- **Extension extractions**:
    - `data/<hash>/mcp_run_ontomops/extraction_*.txt`
    - `data/<hash>/mcp_run_ontospecies/extraction_*.txt`

If they don't exist, we either:

- run the baseline pipeline once (normal mode), then run ablation; or
- do the "strict ablation" variant (rerun extraction with external tools disabled).

Execution runbook (what we'll run once implemented)

A) Fast ablation (reuse intermediates; external tools disabled during KG building + downstream)

- Example (single DOI hash):
  - `python generic_main.py --config configs/pipeline.json --hash <HASH> --start-step main_kg_building --disable-external-mcp --ablation-name Ablation_test_One_external_disabled`

Expected:

- earlier steps are skipped (because we start at KG building)
- KG-building outputs are written under:
  - `data/<hash>/ablation/Ablation_test_One_external_disabled/...`
- logs show that `mcp_tools` lists have been filtered to exclude the external tools.

B) Strict ablation (rerun extraction too; slower)

- Start from `main_ontology_extractions` (and optionally `extensions_extractions`) with the same flags.
- This requires:
  - output-isolated paths for extraction prompts/responses/hints (so we don't overwrite baseline)
  - enforcement of `blocked_mcp_tools` during iter2 extraction (where external tools are typically used)

What "results" we will report for this ablation

We will add a small evaluation script (or notebook) that compares **baseline vs ablation output folders** for each DOI hash:

- **KG artifact presence**
  - number of TTL files produced (per ontology and per entity)
  - missing/empty TTL outputs
- **Graph size**
  - triple count per TTL (rdflib parse + `len(graph)`)
  - total triples aggregated across TTLs
- **Key field coverage**
  - counts of entities with:
    - CCDC numbers present (expected drop when `ccdc` disabled)
    - SMILES/InChI present (expected drop when `pubchem`/`chemistry` disabled)
    - yield / conditions fields (may or may not change)
- **Runtime + failure rate**
  - wall-clock runtime per step
  - number of agent failures / retries

We will summarize the above as a table in this file once the run completes.

Risks / gotchas to address during implementation

- **BaseAgent strictness**: if a tool remains in `mcp_tools` but is blocked/not configured, the run will fail. Filtering is mandatory.

- **Markers**: existing `.main_kg_building_done` etc in the baseline folder must not cause ablation steps to skip. Markers must be written/read from the ablation output root.
- **Global state file**: KG-building steps write `data/global_state.json` for MCP servers; we must ensure ablation runs don't corrupt baseline runs (avoid parallel runs; optionally include run_tag inside global state).
- **Prompt instructions referencing external tools**: prompts may tell the agent to "use PubChem / search". With tools disabled, the agent must fall back to paper-only reasoning. We'll keep this as part of the ablation effect (don't "help" the agent beyond enforcing the tool block).