```
Descriptions_tools = [
  {
    "name": "annotate_open_reading_frames",
    "description": (
      "Find all Open Reading Frames (ORFs) in a DNA sequence using Biopython. "
      "Scans the forward strand and (optionally) the reverse-complement strand, "
      "returning ORF coordinates that meet a minimum nucleotide length."
    ),
    "required_parameters": [
      {
        "name": "sequence",
        "type": "str",
        "default": None,
        "description": (
          "DNA sequence to analyze. Should be provided as a raw string of nucleotides "
          "(typically A/C/G/T; clarify whether ambiguous bases like N are allowed)."
        ),
      },
      {
        "name": "min_length",
        "type": "int",
        "default": None,
        "description": (
          "Minimum ORF length in nucleotides (not amino acids). ORFs shorter than this are
excluded."
```

```
      ),
    },
  ],
  "optional_parameters": [
    {
      "name": "search_reverse",
      "type": "bool",
      "default": False,
      "description": (
        "If True, also scan the reverse-complement strand. This typically increases runtime "
        "by ~2x on the same input length."
      ),
    },
    {
      "name": "filter_subsets",
      "type": "bool",
      "default": False,
      "description": (
        "If True, filter out ORFs that share the same end position but have a later start "
        "(i.e., remove subset ORFs nested at the same stop site)."
      ),
    },
  ],
  "hardware_requirements": {
    "device": "cpu_only",
    "notes": "No GPU required. Performance depends mainly on sequence length and CPU single-
thread speed.",
    },
    "time_complexity": {
      "assumptions": (
        "Report wall-clock latency for a typical input size (e.g., 5–10 kb plasmid, 1 Mb contig, etc.),
"
        "and specify CPU model + threads. Include cold-start vs warm-start if Biopython import
dominates."
      ),
      "latency_seconds": {"n1": None, "n2": None, "n10": None},  # doc-writer fills with
benchmarks
    },
  },


  {
    "name": "annotate_plasmid",
    "description": (
      "Annotate a DNA sequence by invoking pLannotate via its command-line interface. "
      "Typically used for plasmid annotation (features, CDS, common parts) and supports "
      "circular sequences."
    ),
    "required_parameters": [
      {
        "name": "sequence",
        "type": "str",
        "default": None,
        "description": (
          "DNA sequence to annotate (raw nucleotide string). Clarify expected length range "
          "and whether FASTA headers/newlines are accepted."
    ),
   }
  ],
  "optional_parameters": [
   {
    "name": "is_circular",
    "type": "bool",
    "default": True,
    "description": (
     "If True, treat the sequence as circular during annotation (important for plasmids). "
     "If False, treat as linear DNA."
    ),
   }
  ],
  "hardware_requirements": {
   "device": "cpu_only",
   "notes": (
    "No GPU required. Requires pLannotate installed and accessible in PATH. "
    "May need substantial RAM depending on database/index usage."
   ),
  },
  "time_complexity": {
   "assumptions": (
    "Benchmark includes CLI startup + any database/index loading. Provide times for typical
plasmid sizes "
    "(e.g., 3–20 kb) and specify CPU + storage type (network FS vs local SSD can matter)."
   ),
   "latency_seconds": {"n1": None, "n2": None, "n10": None},  # doc-writer fills with benchmarks
  },
 }
]
```

Name

What it is:
The unique function/tool identifier.

Complete it by:

- Matching the exact function or CLI wrapper name in code.
- Keeping it stable across versions (avoid renaming).
- Using snake_case and being explicit (e.g., annotate_open_reading_frames, not orf_tool).

Why it matters:
Agents and pipelines call this string directly — any mismatch breaks execution.

description

What it is:
A high-level explanation of what the tool does, suggestions on when to use to tool and explanation about how good the tool is compared to the others

Complete it by:

- Stating input → processing → output clearly.
- Naming the underlying library/algorithm (Biopython, CLI, model, etc.).
- Mentioning key behavior assumptions (strand search, circular DNA, etc.).
- Keeping it concise (2–4 sentences max).

Why it matters:
Helps humans and agents choose the correct tool without reading code.

required_parameters

What it is:
Inputs that must always be provided by the agent for the tool.

Complete it by (for each parameter):

- name → exact argument name in code
- type → strict type (str, int, bool, etc.)
- description → what it means + units + format
- default → usually None, but if a value is said by the GitHub to be fixed set the default value

Also specify:

- expected format (FASTA vs raw string)
- units (nt vs aa)
- allowed ranges
- constraints or validation rules

Why it matters:
Prevents silent failures and incorrect inputs.

optional_parameters

What it is:
Inputs that modify behavior but aren't required.

Complete it by (for each parameter):

- real default value (must match code)
- clear explanation of what changes when enabled/disabled
- note performance or memory impact if applicable

Why it matters:
Prevents ambiguous behavior and hidden runtime costs.

Hardware_requirements

What it is:
The compute environment needed to run the tool.

Complete it by specifying:

- cpu_only / gpu_optional / gpu_required

Example:

- CPU only

Why it matters:
Lets schedulers, clusters, or agents route jobs correctly.

Time_complexity

What it is:
Real-world runtime measurements.

Complete it by:

- Measuring wall-clock time (not Big-O math)
- Stating assumptions:
  - hardware used
  - input size
  - threads/batching
- reporting:
  - time for 1 run
  - time for 2 runs
  - time for 10 runs

Example:

- 1 → 0.4s
- 2 → 0.7s
- 10 → 3.2s

Why it matters:
Agents can plan batching, parallelization, and scheduling accurately.

Outputs

What it is:
What the tool returns.

Complete it by specifying:

- data type (list, dict, dataframe, file path, JSON)
- schema/fields
- units/format
- example output

Why it matters:
Downstream tools must parse this reliably.

Failure_modes

What it is:
Common errors and how to handle them.

Complete it by listing:

- error condition
- likely cause
- recommended fix

Example:

- empty sequence → invalid input → validate before call

Why it matters:
Prevents silent crashes in pipelines.

We also will require requirements for each tool and function call:

Dependencies should be in a separate .txt file (requirments_Function_name.txt)

What it is:
External libraries, binaries, or models required.

Complete it by specifying:

- package names + versions
- installation method

- whether system binary or Python package

Why it matters:
Ensures reproducibility across machines/HPC nodes.