Node System Development Checklist

> 这是一份**待开发**的功能/改进项清单，涵盖了我们在 `alphaflow-nodes` 系统中所需处理的主要问题、以及对应的开发要点。请使用下方复选框来追踪每个功能的完成进度。

---

1. Parameter Validation & Type Safety

- [ ] **Param Struct + Deserialize**
  - 为每个节点创建 `*_params.rs` 文件，使用 `#[derive(Deserialize)]` 强类型解析。
- [ ] **`validate()` 方法**
  - 在 param struct 中加入校验逻辑(非空、取值范围等)。
- [ ] **通用解析辅助**
  - （可选）在 `shared/validation.rs` 或类似模块中编写 `parse_params<T>` 函数，用统一方式从 `serde_json::Value` 提取并报错。
- [ ] **(可选) Workflow-level 检查**
  - 在 orchestrator 里，对节点的"静态"参数做一次性验证，提前报错。

---

2. Handling Diverse Input/Output Data

- [ ] **MVP: 使用 `serde_json::Value`**
  - `NodeOutput` 里先保留 `data: Value`，让节点灵活返回各类 JSON。
- [ ] **(可选) Node Output Schema**
  - 如果未来要在 orchestrator/前端自动化解析输出结构，考虑定义"output_schema()"或在文档中说明。
- [ ] **(可选) Typed Pipeline**
  - 如果对编译期类型安全要求高，可在后期引入"typed transforms"或 `NodeData` enum 来强化下游解析。

---

3. Resource / Folder Management

- [ ] **节点一级目录**
  - 确保节点都在 `src/<node>/` 下（如 `openai/`, `http/`, `slack/`）。
- [ ] **Hierarchical Grouping**
  - 当节点数增多后，按功能再做二级目录（如 `ai/openai/`）。
- [ ] **Feature Flags**
  - 在 `Cargo.toml` 中配置 `[features]`，让使用者可只编译需要的节点。
- [ ] **Docs & Index**
  - 在 `lib.rs` 或 `README.md` 维护节点文件夹清单，方便查找。

---

4. Repeated or Similar Logic

- [ ] **Shared Utility Modules**

    - 在 `shared/` 中编写 `http_utils.rs`, `ai_utils.rs` 等，提供通用函数（如 `http_request`, `call_openai_completion`）。
- [ ] **Node Templates or Macros**
    - 如果大量节点都是“HTTP GET -> parse -> output”，可用 Rust macro/模板模式减少重复。
- [ ] **Validation Helpers**
    - 若校验模式反复出现，可在 `shared/validation.rs` 放常见检查函数（eg. `ensure_range()`, `is_non_empty()`）并在各节点调用。

---

5. Concurrency & Performance

- [ ] **Async**
    - 确保节点内所有外部调用(HTTP, AI)都是真正异步 (`reqwest::Client`, async/await)。
- [ ] **Worker Limits**
    - 在 orchestrator/engine 里控制并发数量。
- [ ] **Rate Limiting** (OpenAI, Slack, etc.)
    - 在 `shared::ai_utils.rs` 或 `shared::http_utils.rs` 中引入令牌桶 / semaphore 做速率限制。
- [ ] **Monitoring**
    - 考虑在 engine 中暴露 metrics，用来监控任务队列是否堆积。

---

6. UI Integration & Node Descriptions

- [ ] **NodeDescription**
    - 在 `node_type.rs` 里添加 `NodeDescription`、`NodeProperty` 结构，或直接 `fn description(&self)->NodeDescription`.
- [ ] **Minimal Field Types**
    - 先支持 `type_ = "string" / "number" / "boolean"` 等；在前端可简单生成表单。
- [ ] **(可选) Regex / min-max**
    - 在 `NodeProperty` 里加额外字段（`regex`, `min`, `max`），并在节点 `validate()` or orchestrator 中检查。
- [ ] **Versioning**
    - 如果节点参数发生重大变更，需要兼容旧版本 workflows，可支持 “version” 字段以便老参数仍能解析。

---

7. Future Plugin / Distribution Approach

- [ ] **Cargo Features**
    - 在 `Cargo.toml [features]` 里声明 `http`, `openai`, `slack` 等特性，允许按需编译。
- [ ] **Multi-Crate**
    - 若要将节点拆成多个 crates (like `alphaflow-nodes-ai`, `alphaflow-nodes-db`)，可预留相关结构；MVP 阶段可先放在同一个。
- [ ] **(可选) Community Node Concept**
    - 若要“社区节点”，可设计“plugin crate”实现 `NodeType` trait 并由主app编译时引入。Rust动态加载(`.so`/`.dll`) 较复杂，暂不推荐在 MVP 中实现。

8. Large-Scale DAG / Workflow Execution

- [ ] **Basic BFS/DFS**
  - 由 orchestrator / engine 处理，节点只需 `execute(...)`.
- [ ] **Error Branching**
  - Node只报告 `Err(NodeError::...)`; orchestrator决定如何处理(走错误分支？重试？)
- [ ] **Stateful Recovery**
  - (In engine) 选择将节点执行状态存入 DB, 允许重启时恢复
- [ ] **Advanced**
  - 仅当你需要 loops, conditionals, partial success, 这部分才需节点配合更多输出（like node sets "status=FAILED"/"error route"）。

结束语

在**`alphaflow-nodes`** 中，你将**依照以上复选项**逐步实现或改进节点系统。可以在实际代码中实现时**勾选每项**，以确保：

1. **Param 强类型解析**+校验(1),
2. **输出处理**与**目录管理**(2,3),
3. **DRY**通用逻辑(4),
4. **异步并发**与**速率限制**(5),
5. **UI描述**(6),
6. **可插件化**(7),
7. 以及**高级工作流**编排(8)等都能覆盖到。

这样，你的Rust Node库就能在后续规模扩大、节点剧增时仍保持**高可维护性**与**易扩展**。祝你开发顺利!