Bridging LLMs and HPC: A Modular MCP-IOWarp Framework for Scalable Task Execution

Gnosis Research Center @ ILLINOIS TECH

Aum Vinodchandra Sathwara Jaime Cernuda Garcia Luke Logan
Anthony Kougkas Xian-He Sun
asathwara@hawk.iit.edu, jcernudagarcia@hawk.iit.edu,
llogan@hawk.iit.edu, akougkas@iit.edu, sun@iit.edu

ILLINOIS TECH

Motivation

- **Different Platform:** LLM pipelines assume cloud VMs, while HPC tasks run via batch schedulers and parallel filesystems.
- **Gaps in tools:** LangChain, OpenAI function calls, and SmartSim solve pieces but lack a complete LLM → HPC bridge.
- **Prompt Overhead:** In-prompt shell commands increase token counts and raise parse-error and timeout rates.
- **Research need:** Researchers require a reliable, auditable link from chat-style planning to HPC execution without rewriting code for each backend.

Proposed Solution

We present **MCP-IOWarp**, a set of dedicated MCP servers that sit between the LLM and the HPC system. Each server implements a single capability and exposes it through a compact JSON-RPC contract. The LLM issues a high-level request; the matching MCP server forwards it to the cluster via the native scheduler or storage API, captures the outcome, and returns a structured JSON reply together with provenance metadata. Because the contract remains unchanged, adopting a new HPC back-end requires only redeploying servers that bind the same MCP methods to that system.

Architecture & Methodology

LLM → MCP client → capability-MCP server(s) → IOWarp / other HPC backend

There are two deployment methods. *Local*—all modules in one process using `stdio` JSON-RPC. *Cluster*—containerised clients communicate via HTTP/SSE with web-based MCP servers.

MCP client serialises and validates requests; *MCP server(s)* translate JSON-RPC to scheduler/storage calls and log provenance; *Backend* executes tasks (IOWarp in our evaluation).

Each invocation records status, latency, and token counts; these logs support Fig. 3's success-rate and variability metrics, where failures originate from schema violations or 5 s backend timeouts.

MCP-IOWarp Tool Example

```python
# Tool to read a file
@mcp.tool(name="read_file", description="Reads the content of a file")
async def read_file(file_path: str) -> list:
    """Read file content with binary support."""
    try:
        normalized = normalize_path(file_path)
        if not is_path_allowed(normalized):
            raise PermissionError("Access denied")

        with open(normalized, "rb") as f:
            content = f.read()

            # Attempt UTF-8 decoding for text files
            try:
                text_content = content.decode('utf-8')
                return [{"type": "text", "text": text_content}]
            except UnicodeDecodeError:
                # Fallback to base64 for binary files
                return [{"type": "text", "text": base64.b64encode(content).decode('ascii')}]

    except Exception as e:
        return [{"type": "text", "text": f"Error: {str(e)}"}]
```

Figure 1: Server-side MCP-IOWarp tool exposing the read file method

MCP-IOWarp Client Execution

```
Query: read txt file with name trial
[Calling tool read_file with args {'file_path': 'trial.txt'}]
Results: {"type": "text", "text": "Hello from Gnosis Research Center!"}
[Called read_file: {"type": "text", "text": "Hello from Gnosis Research Center!"}]
[Tokens used: 65 (prompt 56, response 9)]
Query: |
```

Figure 2: LLM invoking `read`; MCP-IOWarp returns file content and provenance.

Evaluation Methodology

We tested one LLM under three execution modes: *(1) prompt-embedded commands*, *(2) function-call configuration*, and *(3) MCP–IOWarp JSON-RPC*.

Workload. Five file-I/O tasks (`list directory`, `append file`, `read file`, `create script`, `run script`) were run 10 times each (50 invocations per mode) on a fixed Docker image with a frozen IOWarp backend.

Failure rule. An invocation counts as failed if the JSON schema is rejected, the backend exceeds a 5 s timeout, or the outputs do not conform to the expected format or content.

Metrics captured. Success/Failure rate, median latency, 95th-percent latency band, and tokens per task.

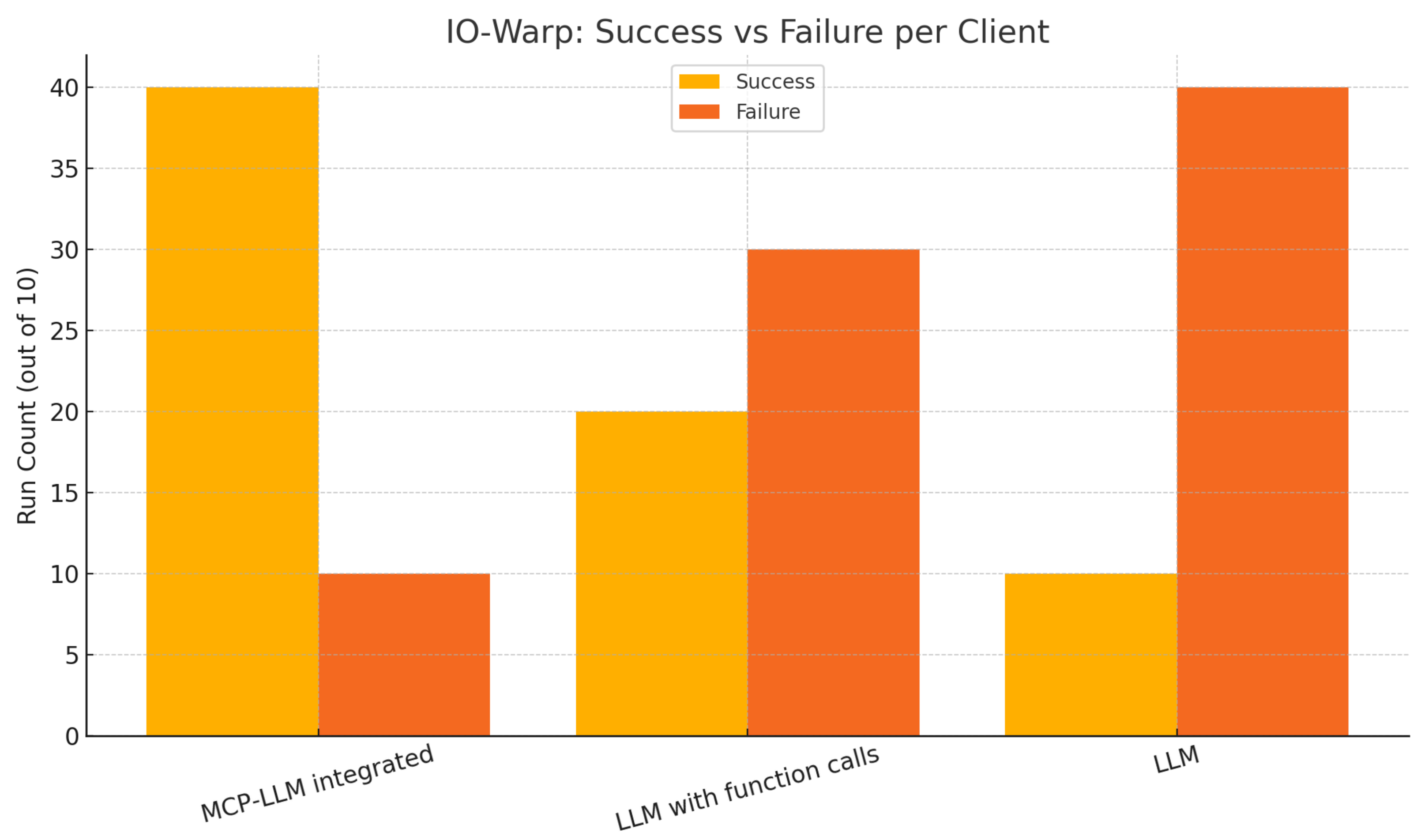

Figure 3: Success and variability across clients.

Conclusion

MCP-IOWarp cleanly separates language-model planning from HPC execution through JSON-RPC servers. Compared with prompt-embedded and function-call modes, it increases task success from 20 %, 40 % to 80 % and reduces median latency by roughly 35 %. Because the JSON contract is backend-agnostic, the same method bindings can target additional HPC systems without prompt changes. Future work will expand the server catalogue, optimise transport for lower latency, and evaluate full scientific workflows.