LINE MCP Server

Getting Started Guide

LINE Development Team

February 22, 2026

1 What is LINE MCP?

LINE exposes its queueing analysis capabilities as a **Model Context Protocol (MCP)** server, allowing Large Language Models such as Claude to build, solve, and analyze queueing models conversationally. Instead of writing code manually, you describe your system in natural language and the LLM calls LINE tools to produce quantitative results.

The MCP server provides 9 tools ranging from quick one-line analyses (no code required) to full custom model building via Python code execution.

2 Setup

2.1 Installation

Install the LINE solver Python package:

```
pip install line-solver
```

Alternatively, clone the repository:

```
git clone https://github.com/imperial-qore/line-solver.git
cd line-solver/python
pip install -e .
pip install mcp numpy pandas
```

2.2 Claude Desktop Configuration

Add the following to your Claude Desktop configuration file (`claude_desktop_config.json`):

```
{
  "mcpServers": {
    "line-solver": {
      "command": "python",
      "args": [
        "/path/to/line-solver/python/mcp_server.py"
      ]
    }
  }
}
```

Replace `/path/to/line-solver` with the actual path to your LINE installation.

2.3 Claude Code Configuration

For Claude Code (CLI), add to your `.claude/settings.json`:

```
{
  "mcpServers": {
    "line-solver": {
      "command": "python",
      "args": [
        "/path/to/line-solver/python/mcp_server.py"
      ]
    }
  }
}
```

Tip

Once configured, you can ask Claude questions like "What is the average queue length for an M/M/1 queue with arrival rate 2 and service rate 3?" and it will automatically use the LINE MCP tools to compute the answer.

3 Quick-Start Tools

These tools require no code—just provide parameter values.

3.1 `analyze_queue` — Open Queue Analysis

analyze_queue

Builds and solves an M/M/k (or M/M/k/N) open queueing model.

Parameter	Default	Description
`arrival_rate`	(required)	Job arrival rate (λ)
`service_rate`	(required)	Service rate per server (μ)
`servers`	1	Number of servers (k)
`queue_capacity`	-1	Total capacity (-1 = infinite)
`scheduling`	FCFS	FCFS, PS, LCFS, or INF
`solver`	MVA	MVA, CTMC, NC, SSA, FLD, or MAM
`format`	text	Output format: `text` or `json`

Example prompt: "Analyze an M/M/1 queue with arrival rate 2 and service rate 5."
The LLM will call:

```
analyze_queue(arrival_rate=2.0, service_rate=5.0)
```

Example output:

```
Model: M/M/1
  arrival_rate (lambda) = 2.0
  service_rate (mu) = 5.0
  servers (k) = 1
```

```
 utilization (rho) = 0.400000
 solver = MVA

 Station Class QLen Tput RespT ResidT ArvR Util
 Queue Class1 0.667 2.0 0.333 0.333 2.0 0.4
```

3.2 `analyze_closed_network` — Closed Network Analysis

analyze_closed_network

Builds and solves a closed queueing network with a Delay (think time) and a Queue.

Parameter	Default	Description
`population`	(required)	Number of circulating jobs (N)
`think_time`	(required)	Mean think time at Delay node (Z)
`service_rate`	(required)	Service rate per server (μ)
`servers`	1	Number of servers (k)
`scheduling`	FCFS	FCFS, PS, LCFS, or INF
`solver`	MVA	MVA, CTMC, NC, SSA, FLD, or MAM
`format`	text	Output format: `text` or `json`

Example prompt: "Analyze a closed system with 10 users, 5 seconds think time, and service rate 2."

```
analyze_closed_network(population=10, think_time=5.0,
                  service_rate=2.0)
```

3.3 `sweep_parameter` — Parameter Sweep

sweep_parameter

Sweeps one parameter over a range and collects performance metrics at each point.

Parameter	Default	Description
`param`	(required)	Parameter to sweep (`arrival_rate`, `service_rate`, `servers`, `population`, `think_time`)
`values`	(required)	Comma-separated values or range notation (`start:stop:step`)
`model_type`	open	`open` or `closed`
Plus all base parameters from `analyze_queue` / `analyze_closed_network`		

Example prompt: "How does queue length change as arrival rate goes from 0.5 to 4.5 in steps of 0.5, with service rate 5?"

```
sweep_parameter(param="arrival_rate",
            values="0.5:4.5:0.5",
            service_rate=5.0)
```

4 Building Custom Models

The `solve_line_model` tool executes arbitrary LINE Python code in a sandboxed environment with `from line_solver import *`, `numpy`, and `pandas` pre-loaded.

4.1 Open Network Example

```
model = Network('M/M/1')
source = Source(model, 'Source')
queue = Queue(model, 'Queue', SchedStrategy.FCFS)
sink = Sink(model, 'Sink')

oclass = OpenClass(model, 'Class1')
source.setArrival(oclass, Exp(1.0))
queue.setService(oclass, Exp(2.0))

model.link(Network.serial_routing([source, queue, sink]))
avg_table = MVA(model).avg_table()
print(avg_table)
```

4.2 Closed Network Example

```
model = Network('ClosedQN')
delay = Delay(model, 'Think')
queue = Queue(model, 'Queue', SchedStrategy.PS)

cclass = ClosedClass(model, 'Users', 5, delay)
delay.setService(cclass, Exp(1.0))
queue.setService(cclass, Exp(3.0))

model.link(Network.serial_routing([delay, queue]))
avg_table = MVA(model).avg_table()
print(avg_table)
```

4.3 Multiclass Example

```
model = Network('Multiclass')
source = Source(model, 'Source')
queue = Queue(model, 'Queue', SchedStrategy.FCFS)
sink = Sink(model, 'Sink')

class1 = OpenClass(model, 'Web')
class2 = OpenClass(model, 'API')

source.setArrival(class1, Exp(1.0))
source.setArrival(class2, Exp(0.5))
queue.setService(class1, Exp(4.0))
queue.setService(class2, Exp(3.0))
```

```
model.link(Network.serial_routing([source, queue, sink]))
avg_table = MVA(model).avg_table()
print(avg_table)
```

4.4 Persistent Sessions

Use `session_id` to persist variables across multiple `solve_line_model` calls:

```
# Call 1: Build model
solve_line_model(code="...", session_id="my_session")

# Call 2: Reuse model from previous call
solve_line_model(
   code="avg_table = NC(model).avg_table()\nprint(avg_table)",
   session_id="my_session"
)
```

5 Advanced Tools

5.1 `compare_solvers` — Multi-Solver Comparison

Runs the same model with multiple solvers and returns a merged results table.

```
compare_solvers(
   code="""
model = Network('M/M/1')
source = Source(model, 'Source')
queue = Queue(model, 'Queue', SchedStrategy.FCFS)
sink = Sink(model, 'Sink')
oclass = OpenClass(model, 'Class1')
source.setArrival(oclass, Exp(1.0))
queue.setService(oclass, Exp(2.0))
model.link(Network.serial_routing([source, queue, sink]))
""",
   solvers="MVA,NC,MAM"
)
```

5.2 `visualize_model` — Mermaid Diagrams

Generates a Mermaid graph definition from a queueing network model, showing nodes, connections, and class information.

```
visualize_model(code="""
model = Network('WebApp')
source = Source(model, 'Arrivals')
lb = Queue(model, 'LoadBalancer', SchedStrategy.PS)
app = Queue(model, 'AppServer', SchedStrategy.FCFS)
sink = Sink(model, 'Departures')
oclass = OpenClass(model, 'Requests')
source.setArrival(oclass, Exp(5.0))
```

```
lb.setService(oclass, Exp(100.0))
app.setService(oclass, Exp(10.0))
P = model.initRoutingMatrix()
P[oclass, source, lb] = 1.0
P[oclass, lb, app] = 1.0
P[oclass, app, sink] = 1.0
model.link(P)
""")
```

The output is a Mermaid diagram that can be rendered by any Mermaid-compatible tool (GitHub, Notion, Mermaid Live Editor, etc.).

5.3 `list_examples` and `get_example_code`

Browse and retrieve source code from LINE's built-in example gallery:

- `list_examples()` — returns a categorized list of available examples
- `get_example_code(name)` — returns the Python source code for a named example (e.g., `"gallery_mm1"`, `"cqn_multiserver"`)

Example prompt: "Show me the code for the multiclass open queueing network example."

5.4 `clear_session` — Session Cleanup

Removes a persistent code session created with `session_id`:

```
clear_session(session_id="my_session")
```

6 Solver Reference

Solver	Best For
MVA	Product-form networks with exponential service (fast, exact)
NC	Closed networks needing exact normalizing-constant analysis
CTMC	Small state-space models; supports non-product-form features
MAM	Phase-type / MAP service distributions (matrix analytic)
SSA	Stochastic simulation; works for any supported model
FLD	Fluid / ODE approximation for transient and large models

Table 1: LINE MCP solver guide

All solvers are available through every MCP tool. The quick-start tools (`analyze_queue`, `analyze_closed_netw` accept a `solver` parameter; for custom models via `solve_line_model`, instantiate the solver class directly (e.g., `MVA(model)`, `CTMC(model)`).

7 Output Formats

All analysis tools accept a `format` parameter:

- **`text`** (default) — Human-readable table output with model summary header.
- **`json`** — Structured JSON with `metadata` and `results` keys, suitable for programmatic consumption.

JSON output example:

```
{
  "metadata": {
    "model": "M/M/1",
    "arrival_rate": 2.0,
    "service_rate": 5.0,
    "utilization": 0.4,
    "solver": "MVA"
  },
  "results": [
    {
      "Station": "Queue",
      "Class": "Class1",
      "QLen": 0.6667,
      "Tput": 2.0,
      "RespT": 0.3333,
      "Util": 0.4
    }
  ]
}
```

8 Tool Summary

Tool	Purpose
`analyze_queue`	Quick M/M/k or M/M/k/N open-queue analysis (no code)
`analyze_closed_network`	Quick closed-network analysis (no code)
`sweep_parameter`	Sweep one parameter over a range and collect metrics
`solve_line_model`	Execute arbitrary LINE Python code (sandboxed)
`compare_solvers`	Run the same model with multiple solvers side-by-side
`visualize_model`	Generate a Mermaid diagram of a queueing network
`list_examples`	Browse available example models by category
`get_example_code`	Retrieve source code of a named example
`clear_session`	Remove a persistent code session

Table 2: All LINE MCP tools