Interfacing MCP with Combinatorial, Convex, and SMT Solvers

Lately I've been working on using MCP beyond just using it for symbolic algebra manipulations, I've been thinking about how to interface large language models with a suite of dedicated solvers for scientific computing exposed as tools to the model. And particularly for automating workflows in physical engineering disciplines.

- Github Source Code
- `git clone https://github.com/sdiehl/usolver.git`

If you're not familiar with MCP, you can read more about it in my previous post or the thousand other breathless thinkpieces on the internet about how it's the greatest thing since sliced bread or something. But in reality it's just a janky way to expose a set of function calls (called tools) to a language model via a JSON RPC interface. Language are models are great at language interpretation and limited "planning", and numerical solvers are great at numerical computation. So let each system do what they do best and synthesize the two to build a hybrid system. MCP is just the glue to bind these scientific Python libraries with the model, essentially the 'cgi-bin' of AI.

The simplest motivation for combining language models with numerical solvers via MCP is that it's *vastly* more efficient for certain problems. Consider the dumb example that you certainly can feed a Sudoku puzzle into one of the frontier reasoning models (o3, deepseek r1, etc) and after spinning for quite some time it can (maybe) solve it by talking to "itself" about trial and error guesses for quite some time. The chain of thought will be this vast sequence of guesses and backtracking. But, you can also feed it into a dedicated numerical or logical solver and it will solve it nearly instantly. The same thing goes for many well-studied classical computer science problems: boolean SAT, job shop, vehicle routing, knapsack problem, shift scheduling, travelling salesman etc. There are a lot of pre-existing efficient solvers that research groups have been building and tuning for decades.

As a less theoretical example, consider a common chemical engineering challenge: specifying the design parameters for a processing plant, focusing on line sizing and the associated fluid dynamics. In a typical day to day workflow engineers must determine optimal pipe diameters and requisite pump specifications to achieve target output flow rates for various chemical products. This task is essentially a system of coupled, often non-linear, governing equations encompassing fluid properties (density, viscosity), mass and energy balances across the network, and pressure drop calculations.

Any design must adhere to operational limits, such as maximum allowable pressures within each pipe segment to ensure structural integrity and safety, minimum flow rates to maintain reaction kinetics or prevent settling, and the physical topology of how pipes, valves, and reactors are interconnected. There are dedicated solvers for this, but they are very expensive, and generally suck. Or people write lots of Excel macros and Goal Seek, which I guess works, but there are much better tools.

The bottleneck is simply that the APIs for many of these open source SMT and convex solvers can be, let's say, character-building. If an LLM, piped through MCP, could translate your plain English "what if" into the arcane syntax these solvers speak, suddenly these super-powerful, often byzantine, systems become way less intimidating and democratize access beyond the need for relying on specialized programmers.

Imagine instead, just telling an LLM, via MCP: "Given these target flow rates for product X, using fluid Y with these properties, and ensuring pressure never exceeds Z psi in any line, what are the optimal pipe diameters and pump specs for lines A, B, and C, which are connected in this configuration? Imagine a process engineer who typically spends a significant portion of their week iteratively translating evolving production targets and safety parameters into precise mathematical constraints for solver input, then painstakingly validating the complex outputs for, say, a new heat exchanger network.

With an LLM-solver integration this same engineer could articulate high-level objectives and constraints in natural language—"design a network to cool stream X from 200°C to 50°C using available cold utility Y, minimizing operational cost while ensuring no tube-side pressure exceeds Z"—and have the system automatically formulate the problem, execute the appropriate solver, and present digestible design specifications. This could compress what was previously a multi-day, error-prone exercise as a human constraint generator and checker into minutes, liberating valuable engineering expertise to focus on genuinely creative challenges of their role.

And not just for this one specific field. The dream is to be able to describe all of the constraints for many complex systems (be it a chemical plant, hospital shift planning, or portfolio optimization) in natural language so that the LLM can plan the constraints, delegate to the solver(s), and let the user have a back and forth with about the problem and solution in way that is natural to them. If done well (and we're not there yet) that would be very very powerful.

Then there's more... let's call it forward-looking play: generating synthetic reasoning data. This one's a bit more niche. If we can get an LLM to bulk formulate tricky problems for these solvers, and then learn from the solver's perfectly structured, provably correct answers, we could bootstrap a ton of high-quality training data and then feed that back in as instruction tuning data.

Prototype

But let's start with the easy ones. That's the Stanford cvxopt for convex optimization, Microsoft's z3 SMT solver, and Google or-tools for combinatorial optimization. Just these three packages alone are extremely powerful. Like a seashell, if you look at their source code you can hear the screams of hundreds of PhD students sacrificed on the altar of science to build them.

I've built a prototype of this called USolver which implements the MCP interface to the solvers.

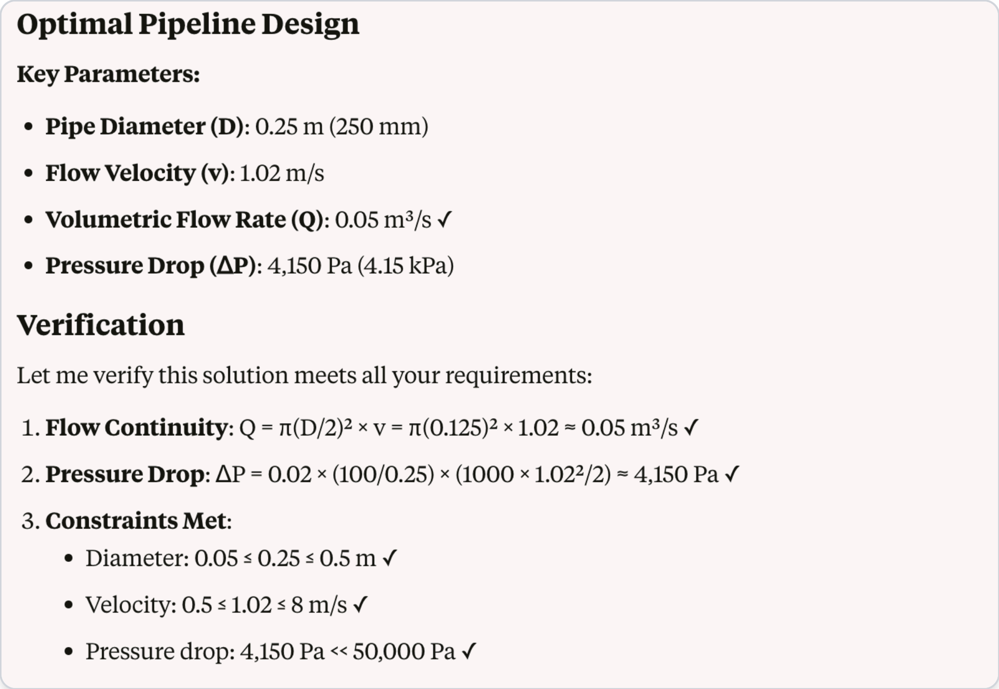
Let's look at some examples. For chemical engineering problems, you can describe a water transport pipeline design task in plain English which gets turned into a Z3 constraint satisfaction problem.

Stephen Diehl

- Index
- Blog
  - Python
  - Mathematics
  - Machine Learning
  - Humor
  - Logic Programming
  - Public Policy
  - Quantitative Finance
  - Functional Programming
  - Compilers
  - Formal Methods
  - Haskell
- Contact Me
- PGP Key
- Github
- Bluesky
- LinkedIn
- RSS

```
Design a water transport pipeline with the followin

* Volumetric flow rate: 0.05 m³/s
* Pipe length: 100 m
* Water density: 1000 kg/m³
* Maximum allowable pressure drop: 50 kPa
* Flow continuity: Q = π(D/2)² × v
* Pressure drop: ΔP = f(L/D)(ρv²/2), where f ≈ 0.02
* Practical limits: 0.05 ≤ D ≤ 0.5 m, 0.5 ≤ v ≤ 8 m
* Pressure constraint: ΔP ≤ 50,000 Pa
* Find: optimal pipe diameter and flow velocity
```



Optimal Pipeline Design

Key Parameters:

- **Pipe Diameter (D)**: 0.25 m (250 mm)
- **Flow Velocity (v)**: 1.02 m/s
- **Volumetric Flow Rate (Q)**: 0.05 m³/s ✓
- **Pressure Drop (ΔP)**: 4,150 Pa (4.15 kPa)

Verification

Let me verify this solution meets all your requirements:

1. **Flow Continuity**: $Q = \pi(D/2)^2 \times v = \pi(0.125)^2 \times 1.02 \approx 0.05$ m³/s ✓
2. **Pressure Drop**: $\Delta P = 0.02 \times (100/0.25) \times (1000 \times 1.02^2/2) \approx 4{,}150$ Pa ✓
3. **Constraints Met**:
   - Diameter: $0.05 \le 0.25 \le 0.5$ m ✓
   - Velocity: $0.5 \le 1.02 \le 8$ m/s ✓
   - Pressure drop: 4,150 Pa << 50,000 Pa ✓

The LLM translates this into a Z3 constraint satisfaction problem, handling all the messy details of flow equations and physical constraints. You get back optimal pipe dimensions and flow parameters that satisfy all your requirements.

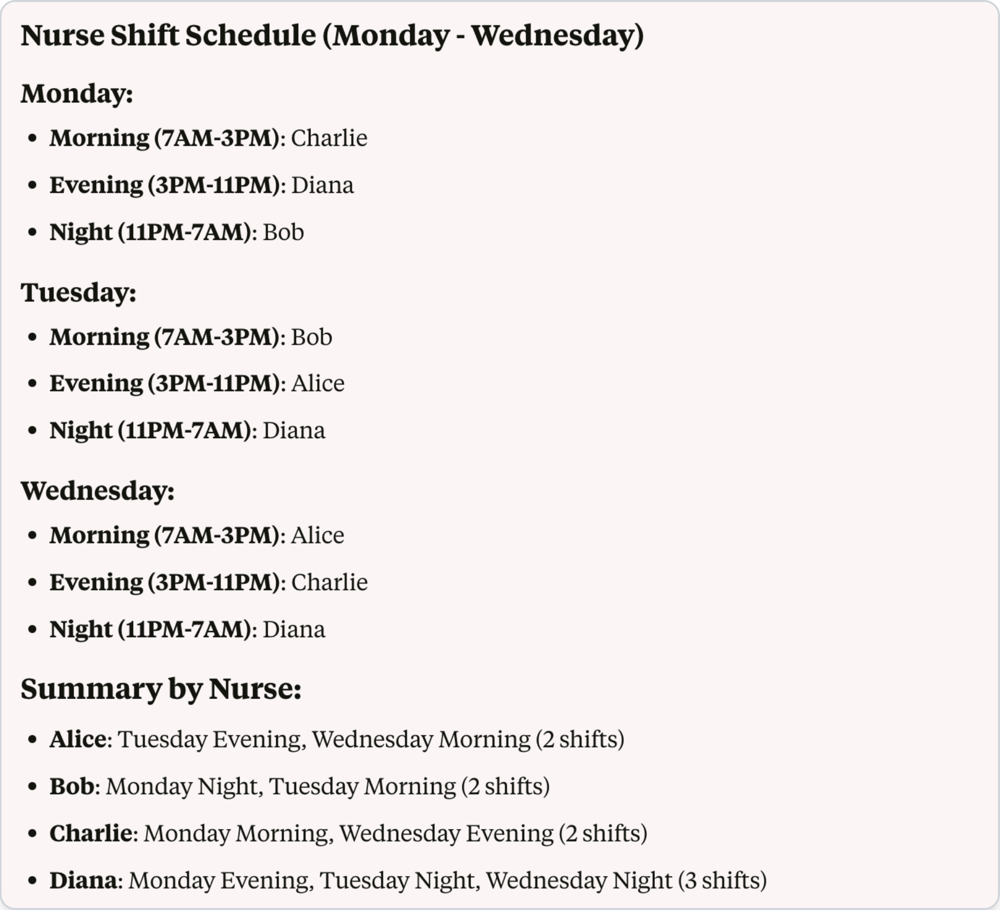
Or take a classic operations research problem like employee scheduling. Instead of wrestling with complex combinatorial optimization syntax, you can just say:

Stephen Diehl

- Index
- Blog
  - Python
  - Mathematics
  - Machine Learning
  - Humor
  - Logic Programming
  - Public Policy
  - Quantitative Finance
  - Functional Programming
  - Compilers
  - Formal Methods
  - Haskell
- Contact Me
- PGP Key
- Github
- Bluesky
- LinkedIn
- RSS

```
Use usolver to solve a nurse scheduling problem wit

* Schedule 4 nurses (Alice, Bob, Charlie, Diana) ac
* Shifts: Morning (7AM-3PM), Evening (3PM-11PM), Ni
* Each shift must be assigned to exactly one nurse
* Each nurse works at most one shift per day
* Distribute shifts evenly (2-3 shifts per nurse ov
* Charlie can't work on Tuesday.
```



Nurse Shift Schedule (Monday - Wednesday)

Monday:

- **Morning (7AM-3PM)**: Charlie
- **Evening (3PM-11PM)**: Diana
- **Night (11PM-7AM)**: Bob

Tuesday:

- **Morning (7AM-3PM)**: Bob
- **Evening (3PM-11PM)**: Alice
- **Night (11PM-7AM)**: Diana

Wednesday:

- **Morning (7AM-3PM)**: Alice
- **Evening (3PM-11PM)**: Charlie
- **Night (11PM-7AM)**: Diana

Summary by Nurse:

- **Alice**: Tuesday Evening, Wednesday Morning (2 shifts)
- **Bob**: Monday Night, Tuesday Morning (2 shifts)
- **Charlie**: Monday Morning, Wednesday Evening (2 shifts)
- **Diana**: Monday Evening, Tuesday Night, Wednesday Night (3 shifts)

But where it gets really interesting is when you chain these solvers together. Here's an example where we optimize a restaurant's operations by combining two different solvers in sequence. First, we use Google's OR-Tools combinatorial optimization to determine the optimal mix and layout of different sized tables within a restaurant's available dining area, maximizing total seating capacity while respecting minimum table requirements and space constraints.

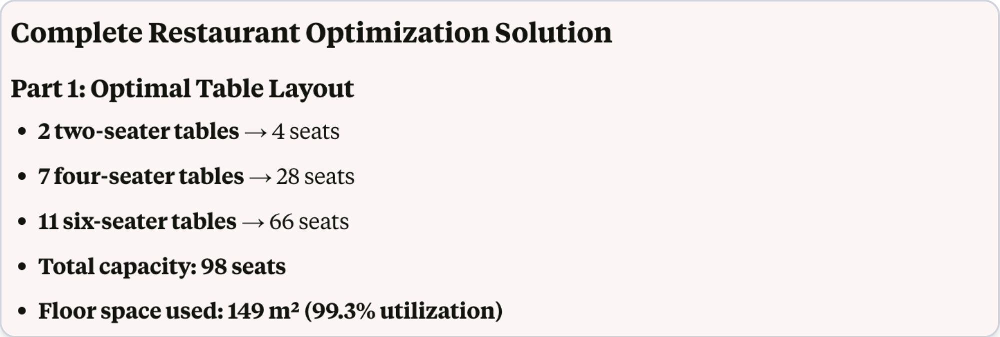
Then, we take that optimal seating capacity and feed it into the cvxopt solver to determine the most cost-effective staffing schedule across a 12-hour day. The staffing optimizer accounts for variable customer demand throughout the day, ensures adequate coverage for the calculated seating capacity, and minimizes labor costs while maintaining service quality and reasonable shift changes. This demonstrates how we can specify a a business problem in plain English, compile it down into a system of constraints, turn those constraints into a system of equations, and then solve it using a chain of numerical solvers with the LLM orchestrating the flow of data and verbalizing the answers!

```
Use usolver to optimize a restaurant's layout and s

* Part 1: Optimize table layout
  - Mix of 2-seater, 4-seater, and 6-seater tables
  - Maximum floor space: 150 m²
  - Space requirements: 4m² (2-seater), 6m² (4-seat
  - Maximum 20 tables total
  - Minimum mix: 2× 2-seaters, 3× 4-seaters, 1× 6-s
  - Objective: Maximize total seating capacity

* Part 2: Optimize staff scheduling using Part 1's
  - 12-hour operating day
  - Each staff member can handle 20 seats
  - Minimum 2 staff per hour
  - Maximum staff change between hours: 2 people
  - Variable demand: 40%-100% of capacity
  - Objective: Minimize labor cost ($25/hour per st
```

And then almost like magic, the LLM coordinates the two solvers and returns the optimal solution. It figures out the optimal table layout and then uses that to schedule the staff. Here's what Claude Desktop generates given the project specification.

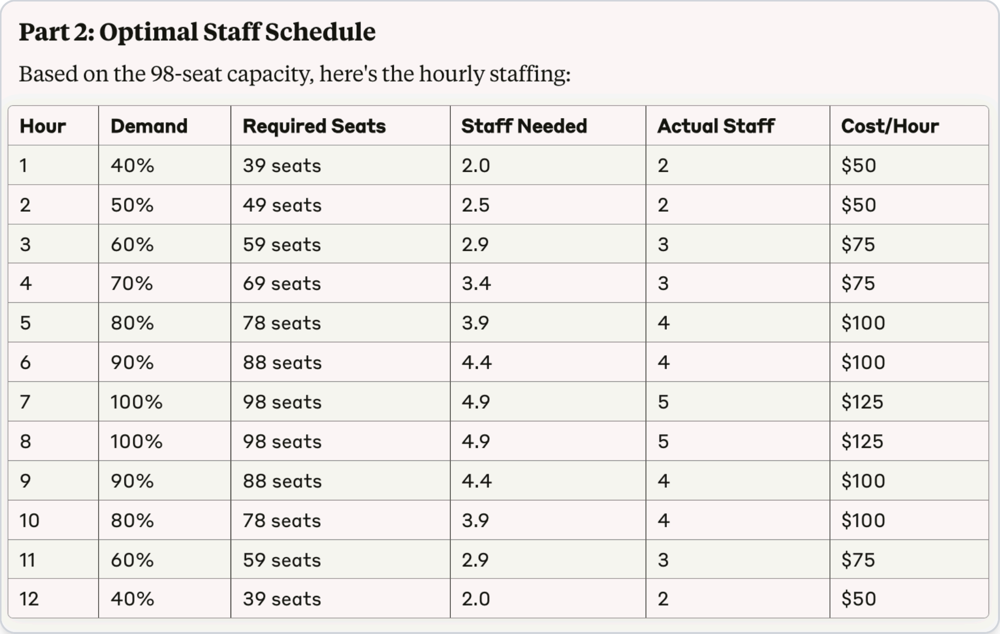


Complete Restaurant Optimization Solution

Part 1: Optimal Table Layout

- **2 two-seater tables** → 4 seats
- **7 four-seater tables** → 28 seats
- **11 six-seater tables** → 66 seats
- **Total capacity: 98 seats**
- **Floor space used: 149 m² (99.3% utilization)**

And then figures out the optimal staffing schedule for that optimal table layout.



Part 2: Optimal Staff Schedule

Based on the 98-seat capacity, here's the hourly staffing:

Hour	Demand	Required Seats	Staff Needed	Actual Staff	Cost/Hour
1	40%	39 seats	2.0	2	$50
2	50%	49 seats	2.5	2	$50
3	60%	59 seats	2.9	3	$75
4	70%	69 seats	3.4	3	$75
5	80%	78 seats	3.9	4	$100
6	90%	88 seats	4.4	4	$100
7	100%	98 seats	4.9	5	$125
8	100%	98 seats	4.9	5	$125
9	90%	88 seats	4.4	4	$100
10	80%	78 seats	3.9	4	$100
11	60%	59 seats	2.9	3	$75
12	40%	39 seats	2.0	2	$50

And it's pretty magical that such a complex problem can be solved in a few seconds with such a simple interface. I'm incredibly bullish about this kind of approach and expanding the set of available tools for the LLM to include many more advanced solvers.

Although don't drink too much of the breathless AI hype Kool-aid just yet. There are some quite real limitations in this approach. Namely, that there is no strict guarantees that LLMs are able to do semantic parsing of the problem with complete accuracy. We can partially mitigate this exposing a computer algebra system and having it check the solutions at each step. There are still mechanisms for transpositions and misinterpretations to slip in.

If this kind of approach becomes mainstream, there could also be benchmarks like SWE-bench for tool-calling of scientific computing packages that could be used to improve model performance upstream.

Stephen Diehl

- Index
- Blog
  - Python
  - Mathematics
  - Machine Learning
  - Humor
  - Logic Programming
  - Public Policy
  - Quantitative Finance
  - Functional Programming
  - Compilers
  - Formal Methods
  - Haskell
- Contact Me
- PGP Key
- Github
- Bluesky
- LinkedIn
- RSS

But at the moment it's still very much a best-effort approach and you would certainly need to have a human-in-the-loop to check the solutions, especially in any kind of safety-critical application. But it might be able to make that human a lot more productive.

The Vision for Universal Solver Interface

Consider the potential of a unified interface that could seamlessly integrate with state-of-the-art solvers across multiple domains with frontier language models via MCP (or whatever the next generation of MCP is):

- **Z3** - SMT solver for constraint satisfaction over booleans, integers, reals, and strings
- **CvxPy** - Framework for convex optimization problems
- **Or-Tools** - Suite for combinatorial optimization and constraint programming
- **MiniZinc** - Discrete optimization solver
- **GAP** - Computational group theory solvers
- **Clingo** - Answer set programming system
- **Flint** - Number theory computations
- **TLA+ / Apalache** - Temporal logic solver
- **Egglog** - Term rewriting and equality saturation engine
- **SymPy** - Comprehensive computer algebra system
- **Souffle** - Datalog-based declarative logic programming framework
- **Mathematica** - Risch-based symbolic integration solver and simplification routines
- **Sage** - Umbrella project for many scientific computing packages

© 2009 - 2025 Stephen Diehl. All rights reserved.