Special Topics: Machine Learning (ML) for Networking

COL867

Holi, 2025

Formal Verification

Tarun Mangla

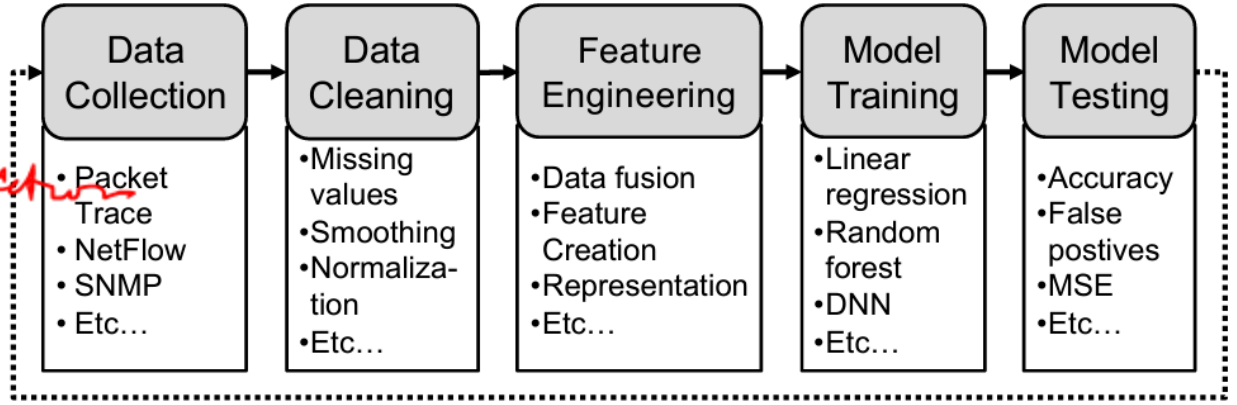
ML for Networks

Module 1: Case studies of specific network learning tasks

Module 2: Task-agnostic automatic ML pipelines for networks

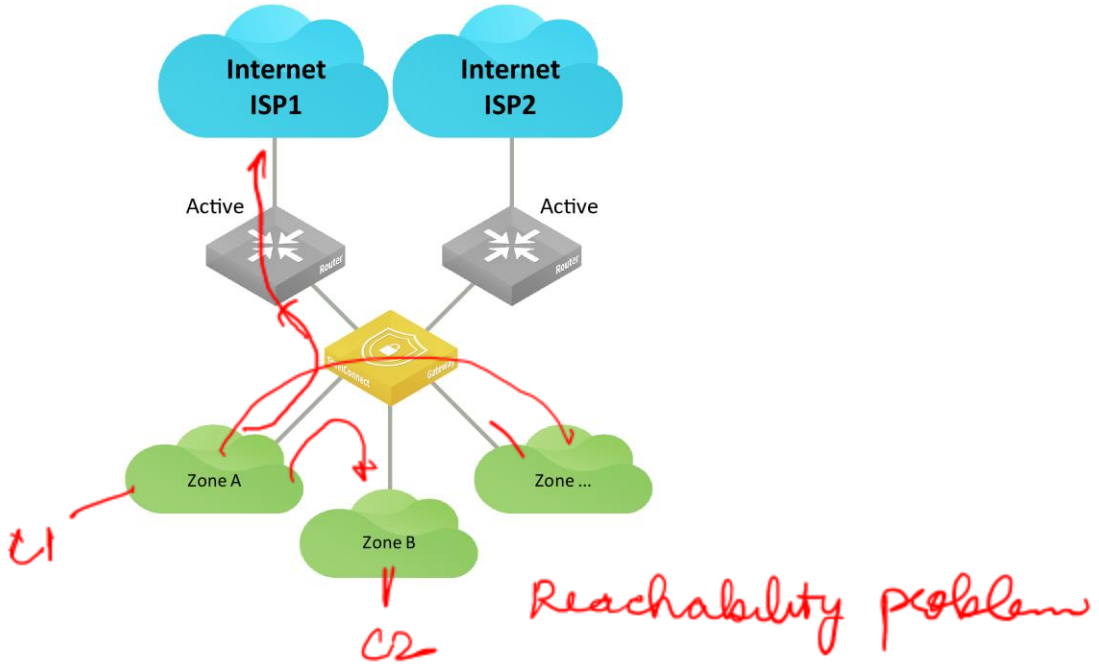
Module 3: Beyond feature engineering and modeling

- Network telemetry
- Robustness
- Explainability
- **Formal verification** / Network verification
- Synthetic data generation



Network Verification

- How to verify the correctness of the network?



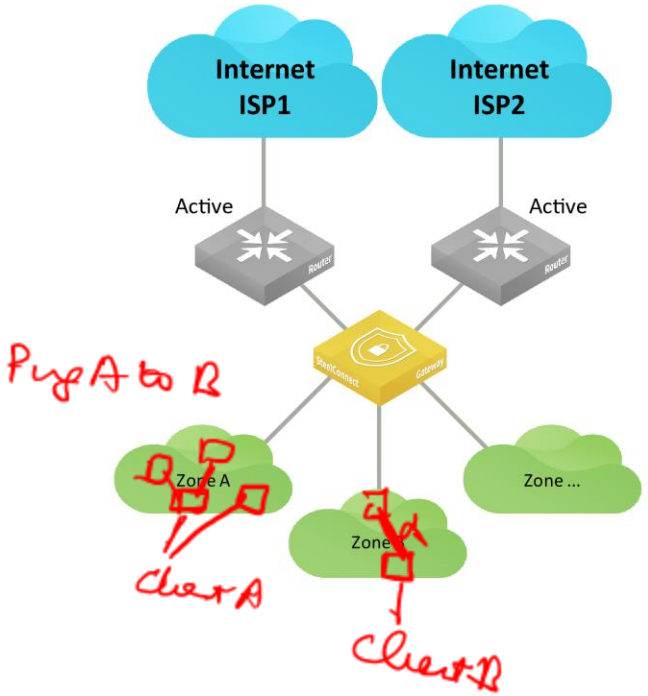
- *A client connected to Zone A can reach a client connected to Zone B*
- *Zone B is not accessible to the public Internet*
- *Clients from Zone A will prefer ISP1 path over ISP2 to connect to the Internet*

→ How to verify network correctness?

Let's consider a concrete example

- Host A in Zone A can reach Host B in Zone B

- Options:
  1. Ping Host B from a node in Zone A. *Limited coverage over space and time*
  2. Regularly monitor the connectivity from different locations in the network. *Still reactive*
  3. Simulate the network behavior in different situations (e.g., link failures, routing inputs). *Proactive but still does not provide guarantee*



The haystack of network behaviors is huge

Large scale

$O(10^3)$ devices
$O(10^4)$ config lines / device
$O(10^6)$ FIB entries / device

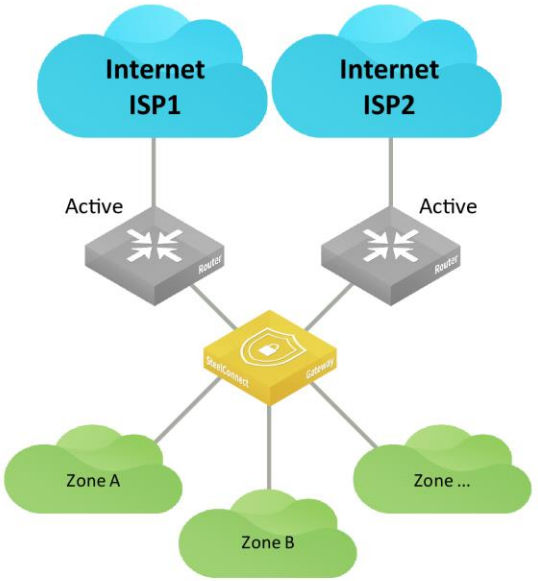
Complex interactions

Distributed routing
Protocol redistribution
Rich route filters

Let’s consider a concrete example

- Host A in Zone A can reach Host B in Zone B

- Options:
  1. Ping Host B from a node in Zone A. *What if a link goes down?*
  2. Regularly monitor the connectivity from different locations in the network. *Still reactive*
  3. Simulate the network behavior in different situations (e.g., link failures, routing inputs). *Proactive but still does not provide guarantee*



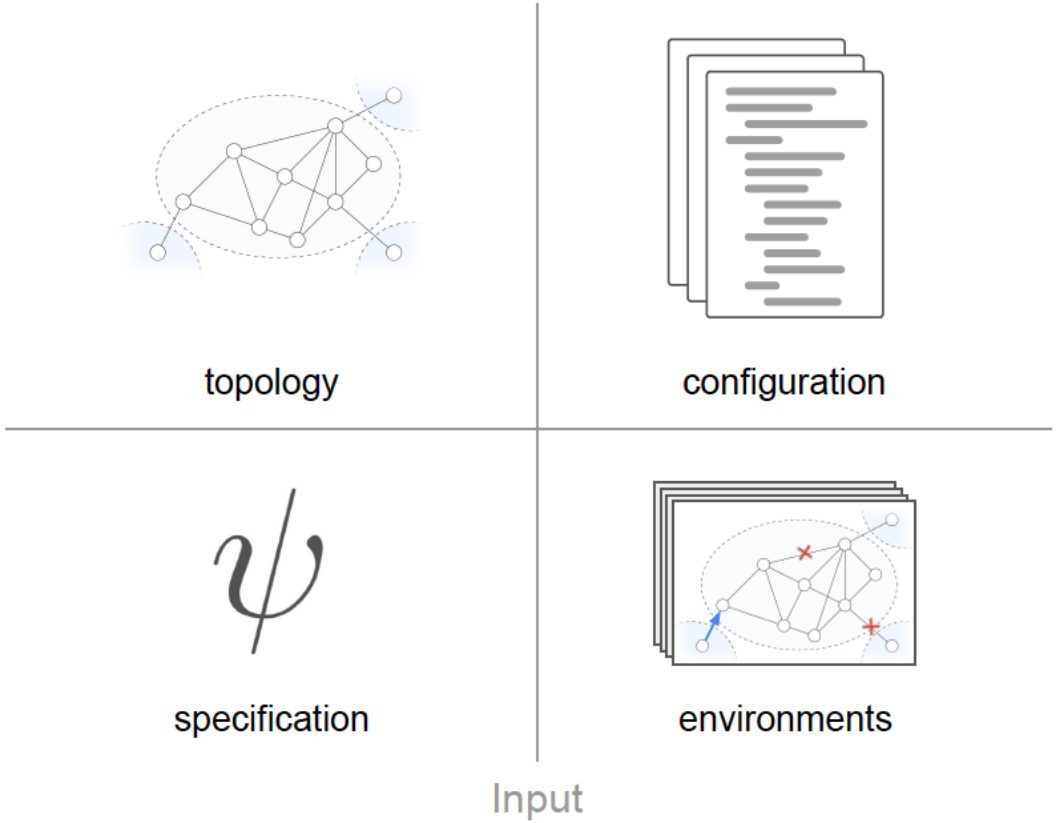
Network Verification

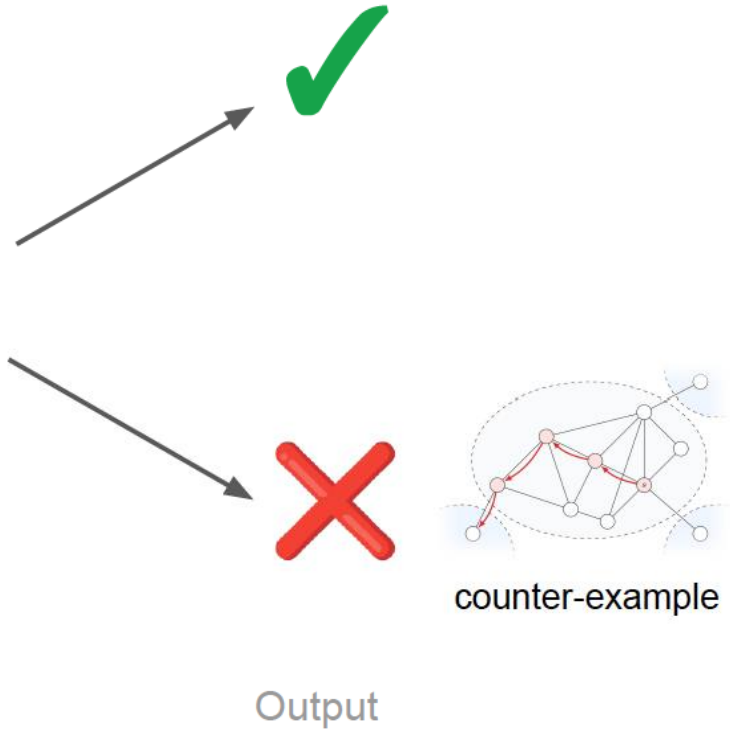
- Verification provides guarantees on a system fulfilling its formal specification in a given universe of environments

- Specification refers to the network's desired
  - Routing/forwarding behavior
  - Security policies
  - Performance (congestion, delay, packet loss)
  - Other properties (e.g., convergence guarantee)

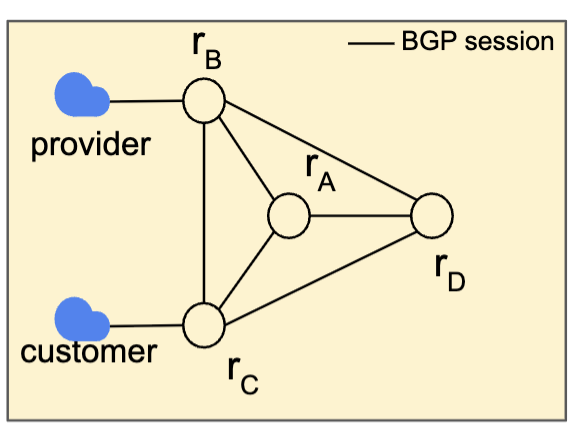
- Given universe of environments describes a set of considered:
  - External routing inputs
  - Failure scenarios
  - Traffic demands

Network Verification

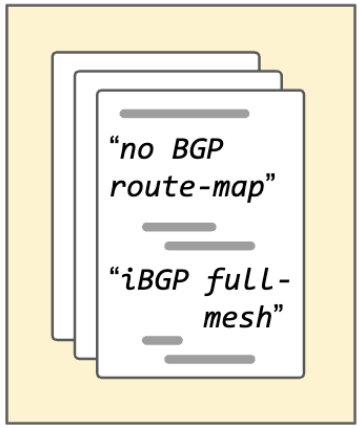




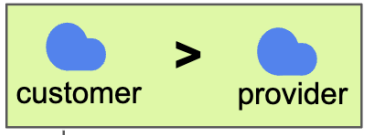
Concrete Example



topology



configuration



specification

environments

"Always prefer the route from the customer.

Input

Can we find a counter-example efficiently and automatically?

counter-example:

	prefix	AS Path length
provider	10.0.0.0/8	1
customer	10.0.0.0/8	2

Output

SAT/SMT

- Propositional logic: Deals with statements that can be either true or false but not both
- Atoms (A) can be either
  - Truth symbols or variables
- Formulas can be either:
  - an atom (A), or
  - Connectives: negation, conjunction, disjunction, implications …
- A solution to a formula F is an assignment of all variables in F such that F evaluates to *true*
- A satisfiability (SAT) solver either returns unsat or sat

SMT

- Satisfiability Modulo Theories (SMT) extends SAT with:
  - Theory of integers, bit vectors, arrays, real numbers, etc
  - Quantifiers

SMT

```
from z3 import *
s = Solver()

x = Int("x")

s.add(x * 2 < 10)
s.add(x > 3)

s.check() # sat
s.model() # [x = 4]
```

- Create symbolic variables of a type (int, real, bool, etc…)
- You can use python operators to build expressions
- You can add multiple assertions to a solver (all of which must be true)
- You can extract the model (i.e., the interpretation that makes all assertions evaluate to `true`.)

Network Verification

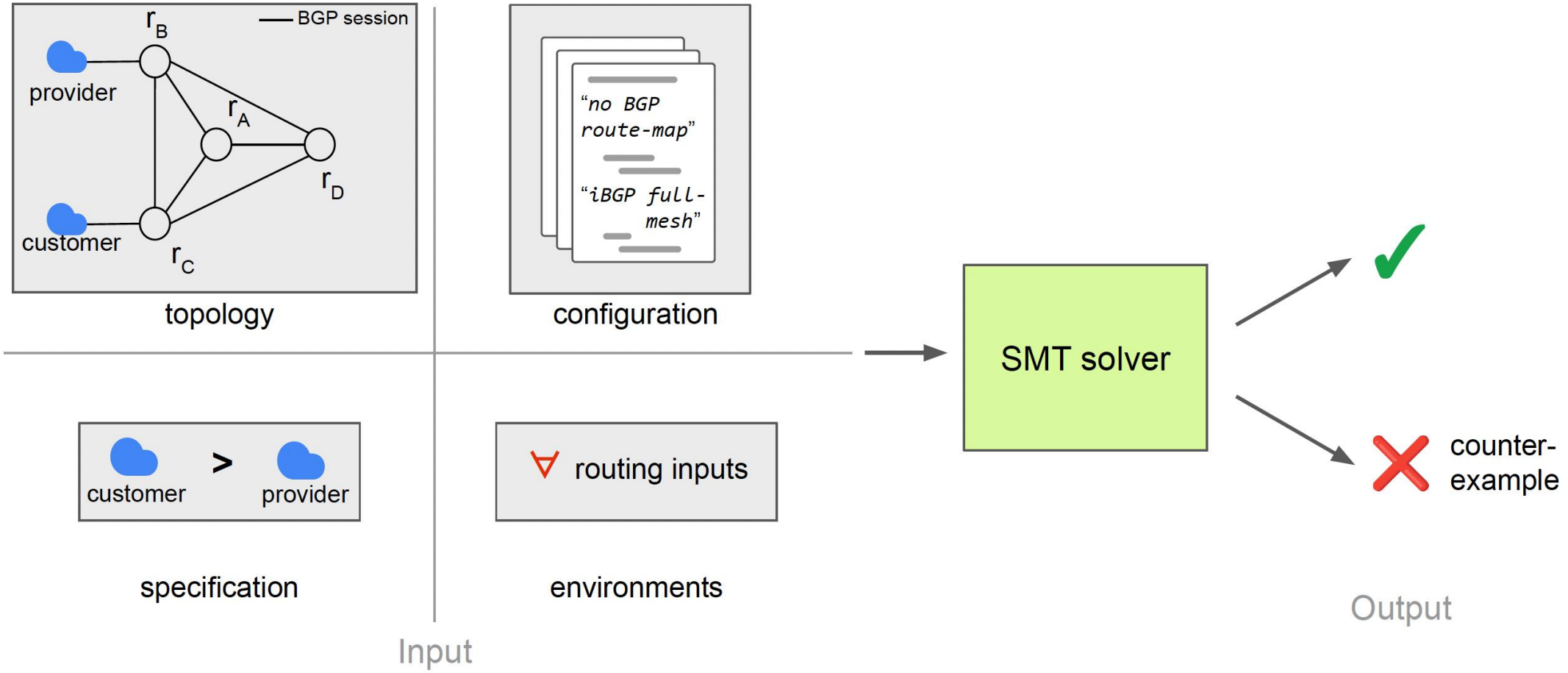
Key idea: Represent the network as a boolean formula which captures the network's outputs given its inputs

- Model the network symbolically, as a combinatorial circuit (as often done in the hardware literature).

- Encode the network's **inputs**/**outputs** in SMT
  - Input: Environment (routing inputs, link failures, etc…)
  - Output: resulting routing state in that environment.
  - Routing semantics: Symbolic formulas to relate inputs to outputs

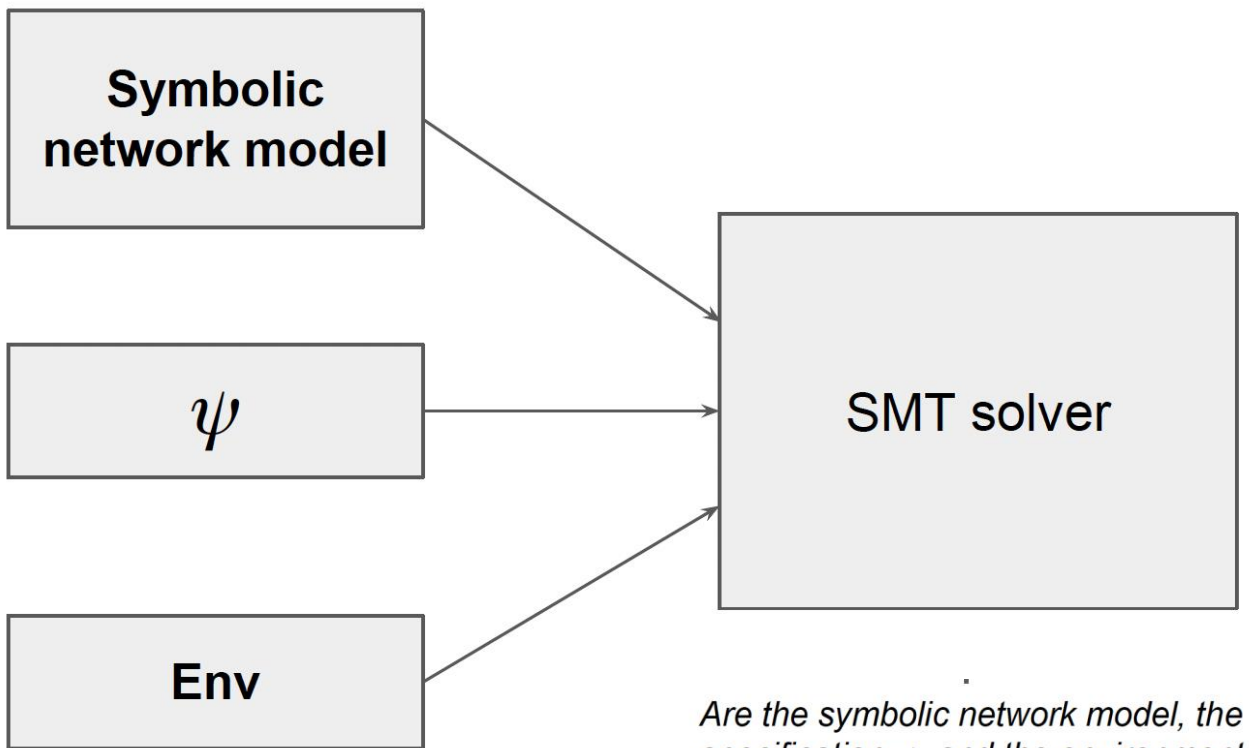
- Given a boolean formula, use theorem provers to assess its satisfiability.
  - Find assignments to the variables that are compatible with the constraints, such that the entire formula is true, if possible.

Using a solver requires to model everything symbolically

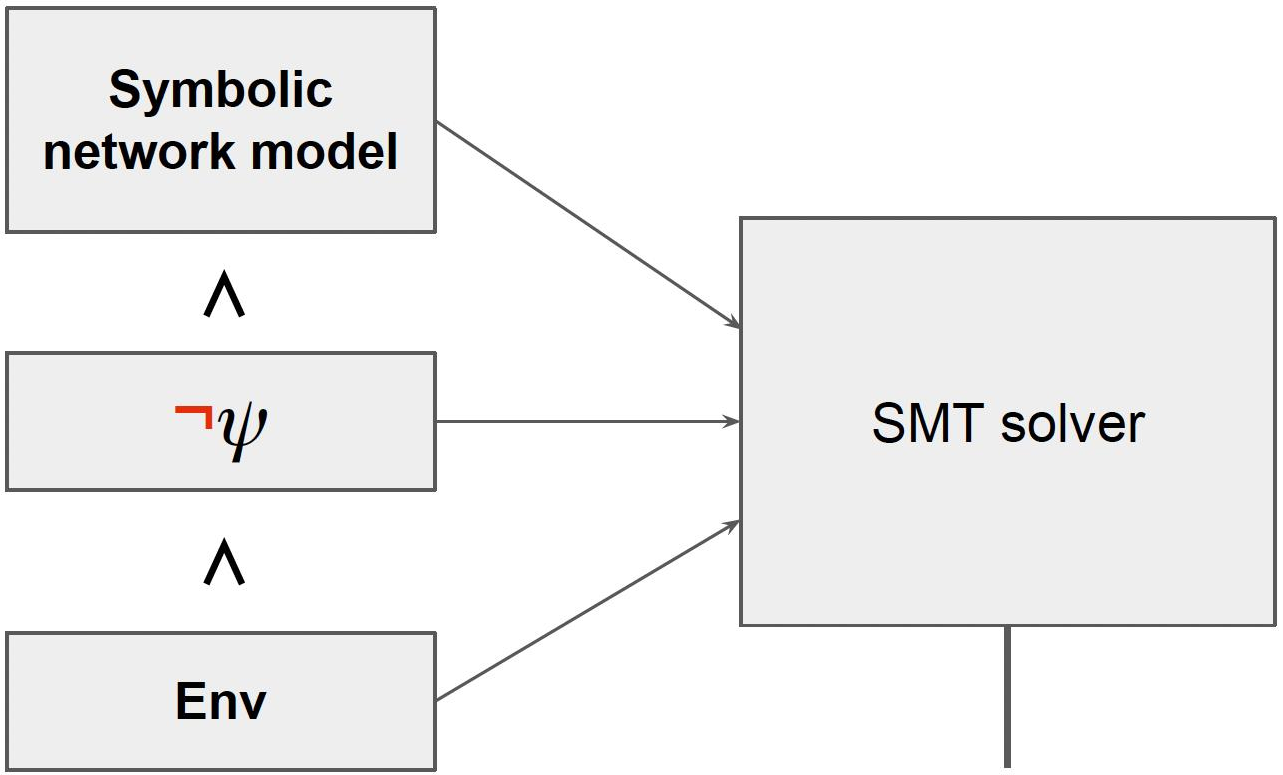


Once we have modeled everything symbolically,
how do check if the specification holds all the time?

in *all* the possible environments



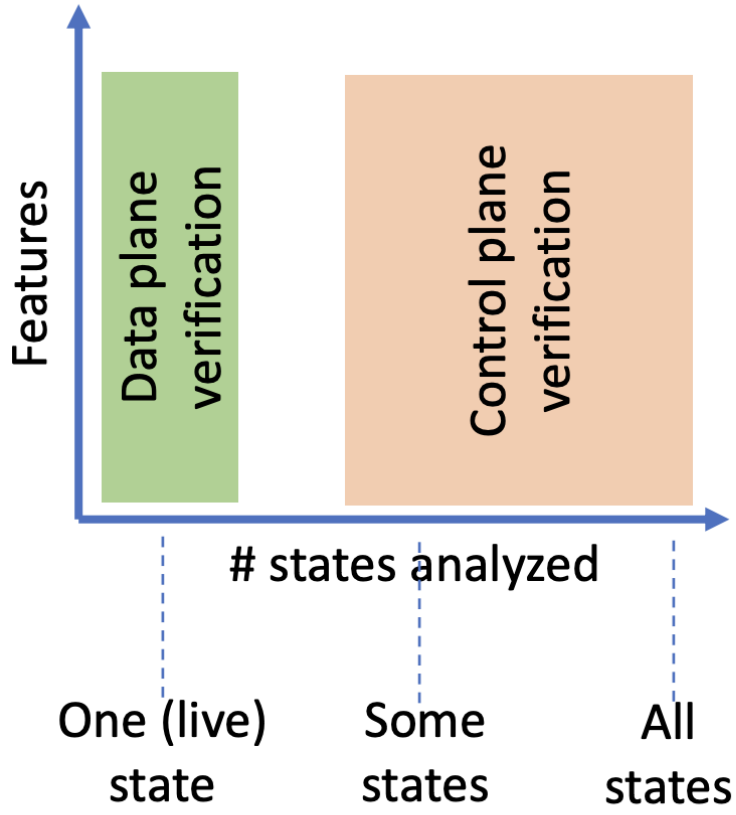
Are the symbolic network model, the specification ψ, and the environment constraints all satisfiable at once?



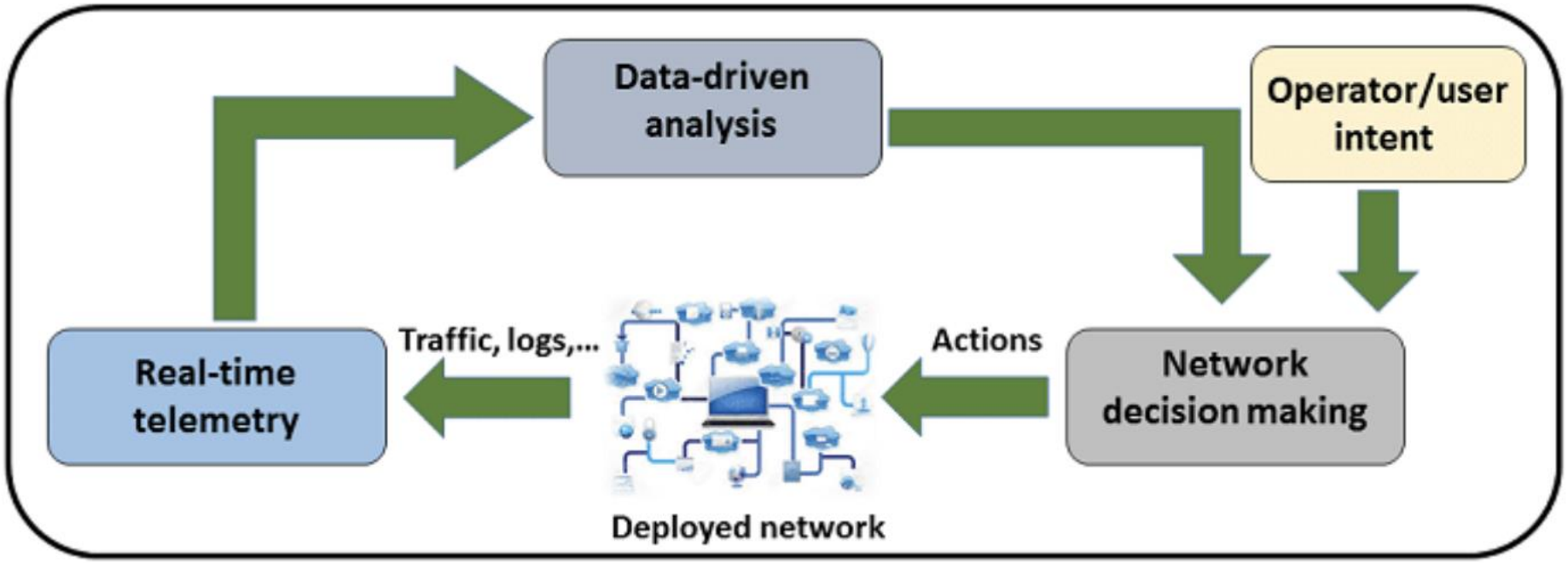
Can the symbolic network model and the environment constraints be satisfied, while the specification ψ is violated?

The 2D space of network verification tools

Shortest-path or policy routing?
Are packet transformed?
Stateless or stateful forwarding?
…



Why is this interesting in context of this course?



Can we use LLMs for generating CORRECT router configurations?

What do LLMs need to Synthesize Correct Router Configurations?

Rajdeep Mondal
UCLA
USA
mondalrajdeep14@ucla.edu

Alan Tang
UCLA
USA
atang42@cs.ucla.edu

Ryan Beckett
Microsoft Research
USA
Ryan.Beckett@Microsoft.com

Todd Millstein
UCLA
USA
todd@cs.ucla.edu

George Varghese
UCLA
USA
varghese@cs.ucla.edu

Abstract

We investigate whether Large Language Models (e.g., GPT-4) can synthesize correct router configurations with reduced manual effort. We find GPT-4 works very badly by itself, producing promising draft configurations but with egregious errors in topology, syntax, and semantics. Our strategy, that we call *Verified Prompt Programming*, is to combine GPT-4 with verifiers, and use localized feedback from the verifier to automatically correct errors. Verification requires a specification and actionable localized feedback to be effective. We show results for two use cases: translating from Cisco to Juniper configurations on a single router, and implementing a no-transit policy on multiple routers. While human input is still required, if we define the *leverage* as the number of automated prompts to the number of human prompts, our experiments show a leverage of 10X for Juniper translation, and 6X for implementing the no-transit policy, ending with verified configurations.

USA. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626111.3628194

1 Introduction

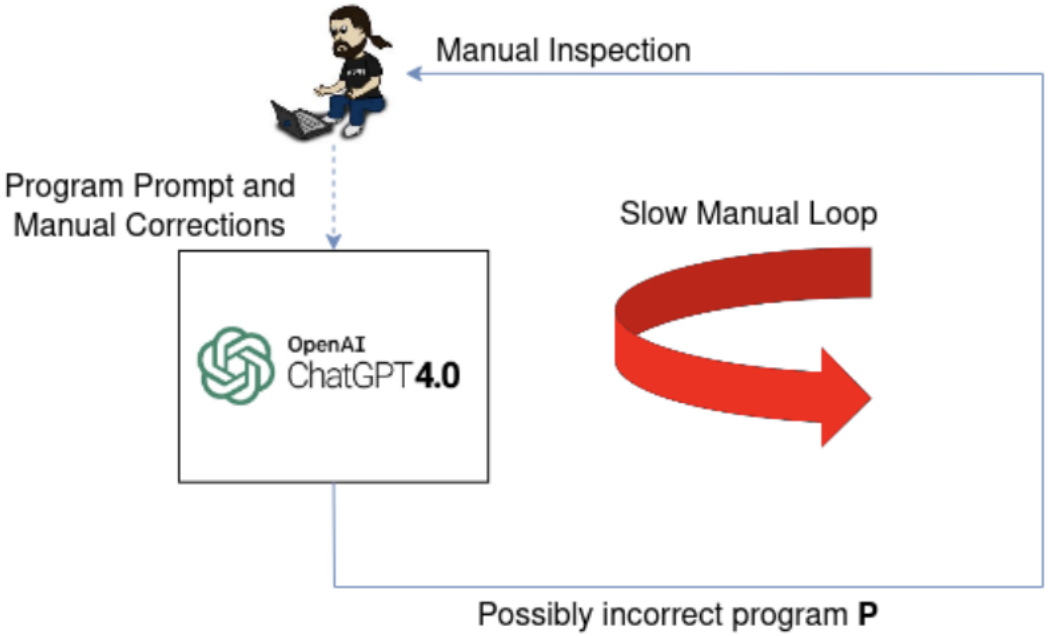
While GPT-4 and other large language models (LLMs) have shown great success in some domains (e.g., writing poems, passing the LSAT) they have been shown to have issues in other domains (e.g,. math, word puzzles) [3]. Language models have had some success in helping users write sequential programs in systems like AlphaCode [10], CoPilot [7], Codex [4] and Jigsaw [8]. They have also been explored as promising assistants for software testing and debugging [13]. Our work investigates code generation by LLMs for a different domain. We examine GPT-4's ability to write router configuration files, traditionally written by humans, that help tune routes and forwarding decisions and are critical for network operation. Our early experiments show that GPT-4 by itself is an "idiot-savant", capable of brilliance but also mak-

Premise of the Paper

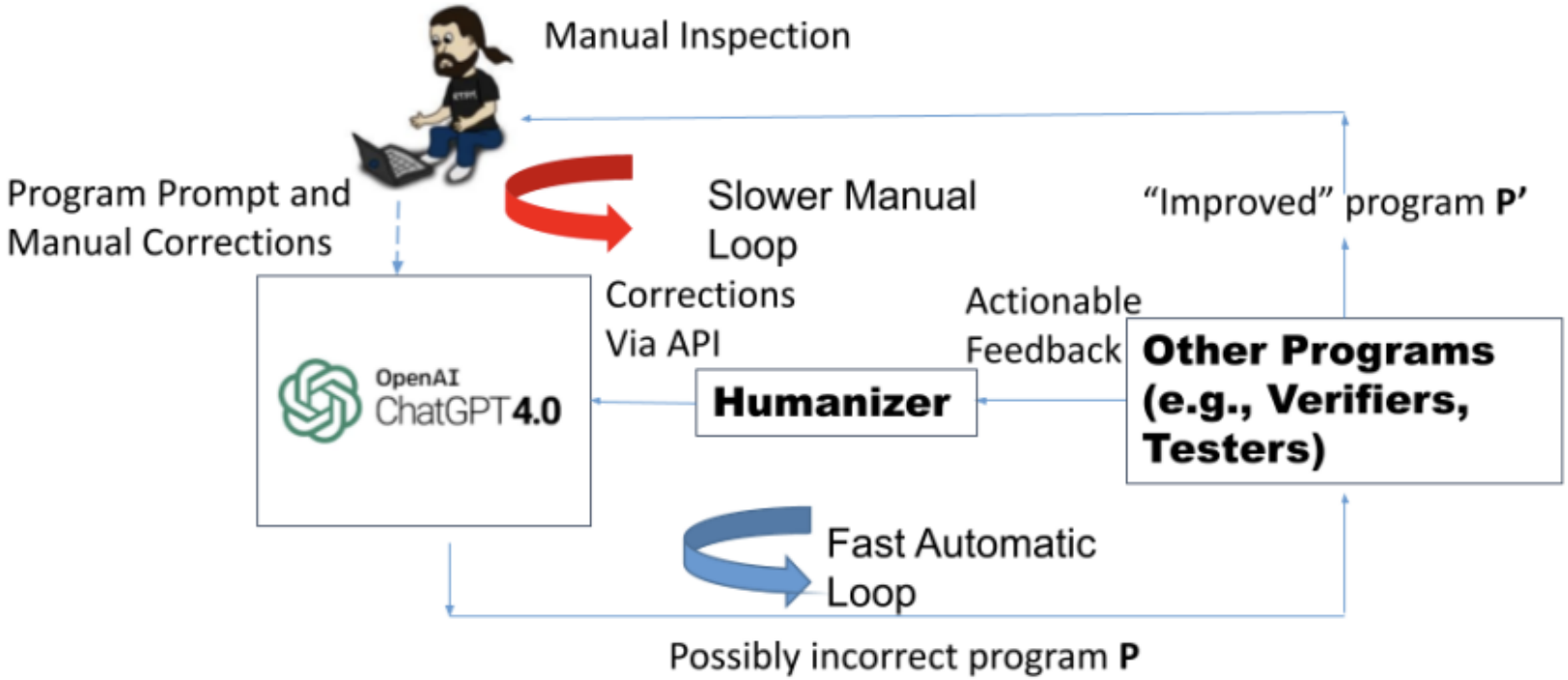
- LLMs are stochastic parrots --> hallucinate all the time
- How do we then use them to generate correct network configuration?
- Specific tasks:
  - Translate a Cisco configuration to equivalent Juniper configuration
  - How do we generate correct translation given that LLMs can hallucinate?

Solutions

- Pair programming using human in the loop



Integrate existing Network Verifiers

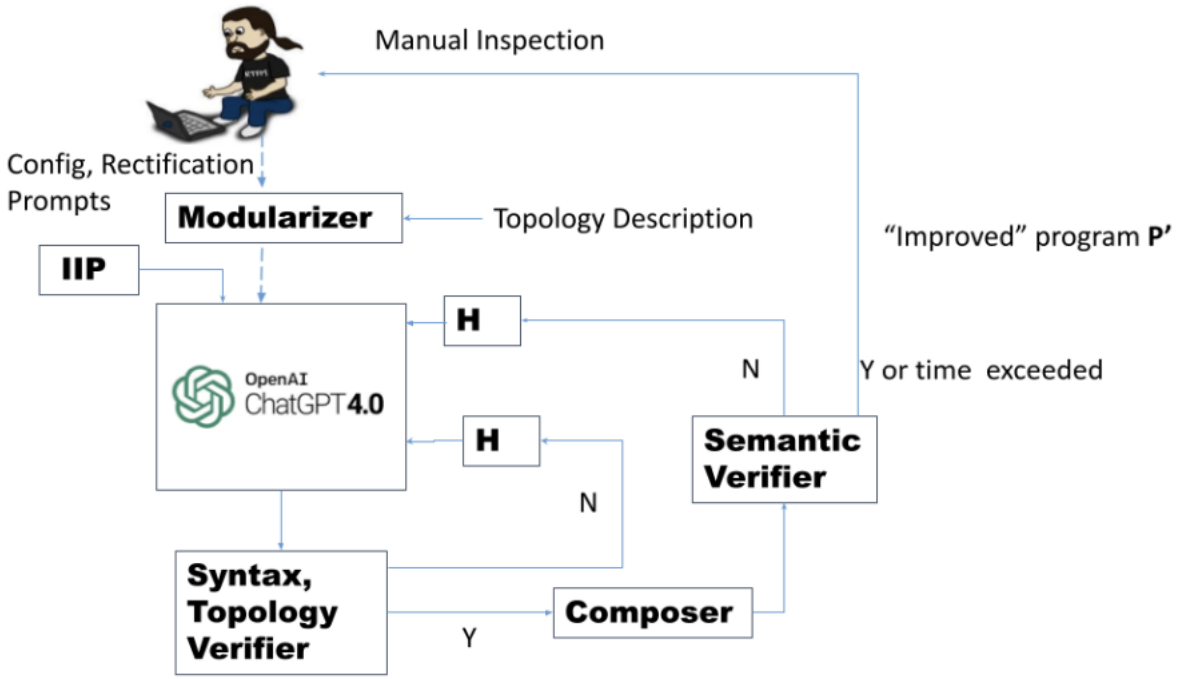


Taxonomy of errors

- Syntax errors
- Structural mismatch
- Attribute differences
- Policy-behavior differences

Type	Generated Prompt
Syntax error	There is a syntax error: '*policy-options prefix-list our-networks 1.2.3.0/24-32*'
Structural mismatch	In the original configuration, there is *an import route map for bgp neighbor 2.3.4.5*, but in the translation, there is no corresponding *route map*
Attribute difference	In the original configuration, *the OSPF link for Loopback0* has *cost* set to *1*, but in the translation, the corresponding *link to lo0.0* has *cost* set to *0*
Policy behavior difference	In the original configuration, for *the prefix 1.2.3.0/25*, the *BGP export policy to_provider for BGP neighbor 2.3.4.5* performs the following action: *ACCEPT*. But, in the translation, the corresponding *BGP export policy to_provider* performs the following action: *REJECT*

Verified Prompt Programming



Evaluation

- *Leverage* (L) as the ratio of the number of automated prompts to the number of human prompts

- Task: Network configuration translation
  - 2 human prompts, 20 automated prompts