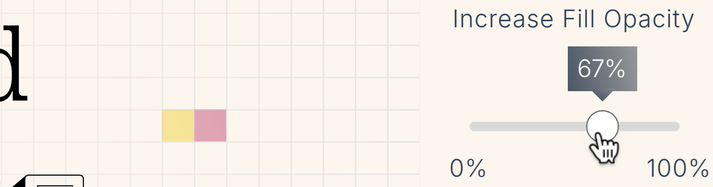
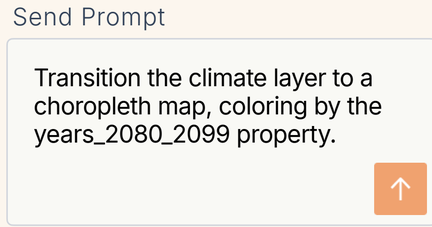
Direct Manipulation and NATURAL LANGUAGE Programming Together at Last?









Parker Ziegler
https://parkie-doo.computer/
peziegler@cs.berkeley.edu



David Minh-Duy Cao
https://cao.sh/
dcao@berkeley.edu



Justin Lubin
https://jlubin.net/
justinlubin@berkeley.edu



Sarah E. Chasins
https://schasins.com/
schasins@cs.berkeley.edu

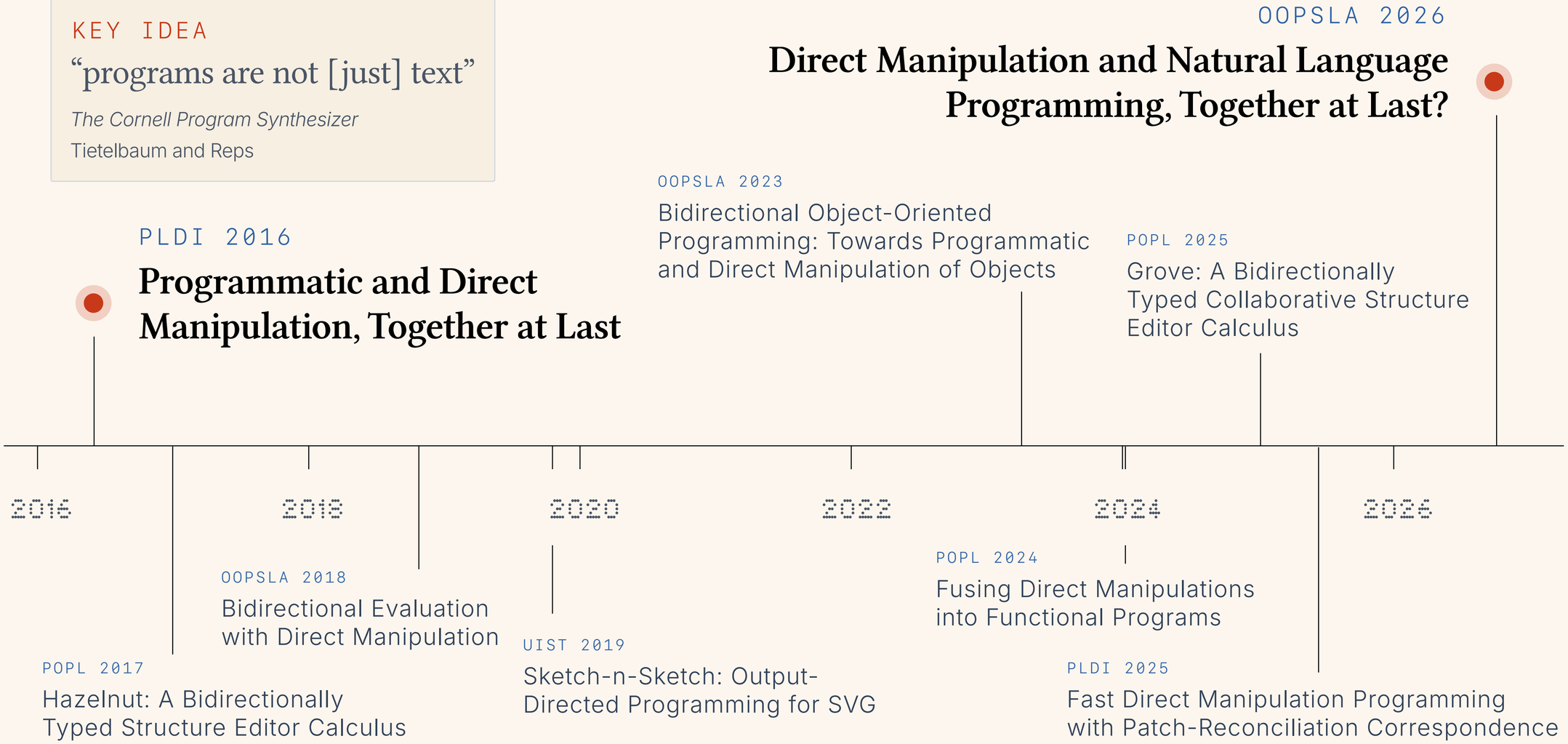
OOPSLA '24
7 Oct 2024

KEY IDEA

“programs are not [just] text”

The Cornell Program Synthesizer

Tietelbaum and Reps



KEY IDEA

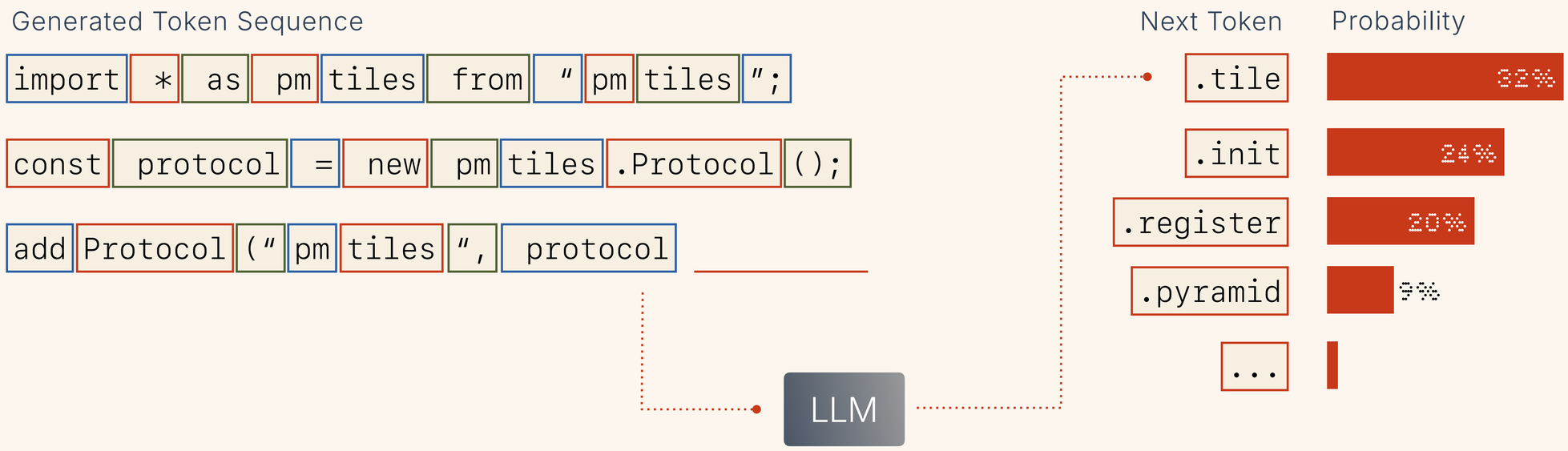
“programs are not [just] text”

The Cornell Program Synthesizer

Tietelbaum and Reps

PROGRAMMING WITH LLMS

Code generation *is* probabilistic, unstructured text generation.



KEY IDEA

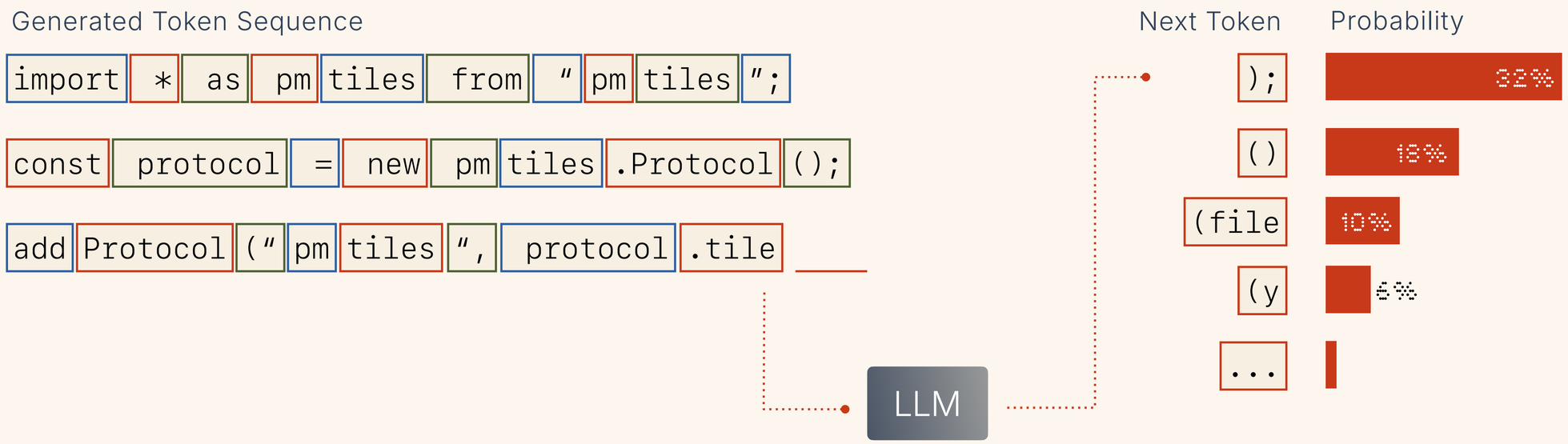
“programs are not [just] text”

The Cornell Program Synthesizer

Tietelbaum and Reps

PROGRAMMING WITH LLMS

Code generation *is* probabilistic, unstructured text generation.



KEY IDEA

"programs are not [just] text"

The Cornell Program Synthesizer

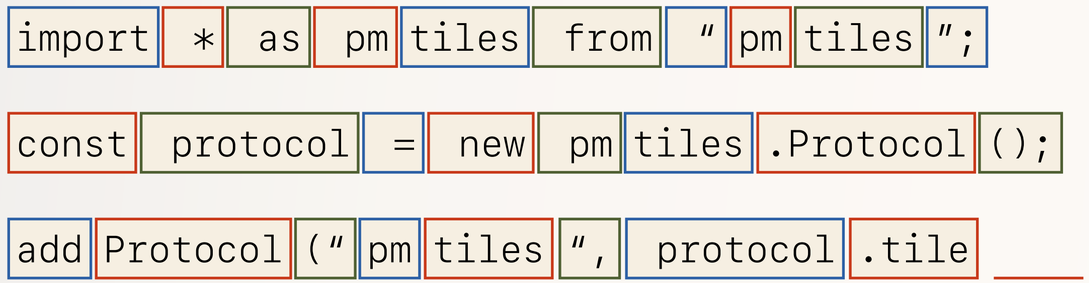
Tietelbaum and Reps

PROGRAMMING WITH LLMS

Code generation *is* probabilistic, unstructured text generation.

Enrich the context window with type and binding information [1]

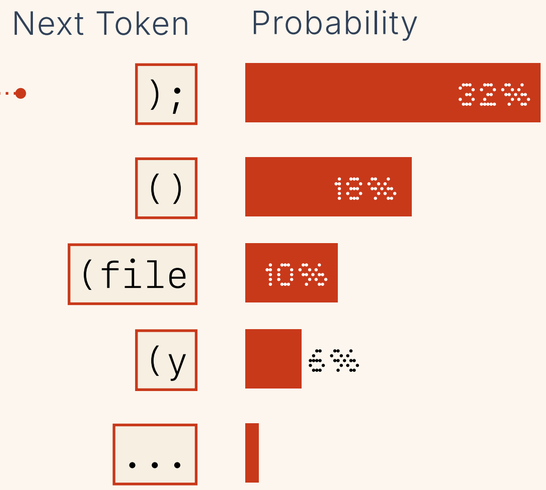
Generated Token Sequence




```
export declare function addProtocol(
  customProtocol: string,
  loadFn: AddProtocolAction
): void;
```

```
type AddProtocolAction = (
  req: RequestParameters,
  abort: AbortController
) => Promise<unknown>;
```

LLM



[1] Blinn, A., Li, X., Kim, J. H. & Omar, C. Statically Contextualizing Large Language Models with Typed Holes.

KEY IDEA

“programs are not [just] text”

The Cornell Program Synthesizer

Tietelbaum and Reps

PROGRAMMING WITH LLMS

Code generation *is* probabilistic, unstructured text generation.

Enrich the context window with type and binding information [1]

Generated Token Sequence

```
import * as pm tiles from “pm tiles”;

const protocol = new pm tiles.Protocol();

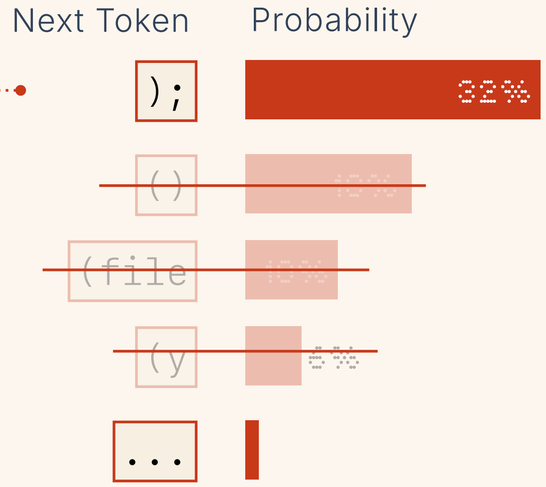
add Protocol(“pm tiles”, protocol.tile
```

```
export declare function addProtocol(
  customProtocol: string,
  loadFn: AddProtocolAction
): void;
```

```
type AddProtocolAction = (
  req: RequestParameters,
  abort: AbortController
) => Promise<unknown>;
```

LLM

Constrain the decoding process with semantic and type constraints [2, 3]



[2] Mündler, N. et al. Type-Constrained Code Generation with Language Models.

[3] Nagy, S., Zhou, T., Polikarpova, N. & D’Antoni, L. ChopChop: A Programmable Framework for Semantically Constraining the Output of Language Models.

Structured Techniques

KEY IDEA

“programs are not [just] text”

The Cornell Program Synthesizer
Tietelbaum and Reps

Automated Refactoring

Structure Editing

Direct Manipulation Programming

Unstructured Techniques

PROGRAMMING WITH LLMS

Code generation *is* probabilistic, unstructured text generation.

Natural Language Programming

THE SYSTEMS QUESTION

How can we develop **unified programming systems** that bridge the gap between these approaches, supporting **multiple editing paradigms in concert**?

THE USER QUESTION

And how would such systems **change the way we program**?

THE SYSTEMS QUESTION

We extended **cartokit** — a **direct manipulation** programming system for interactive maps — with support for **natural language editing in cartokit** DM+NL .



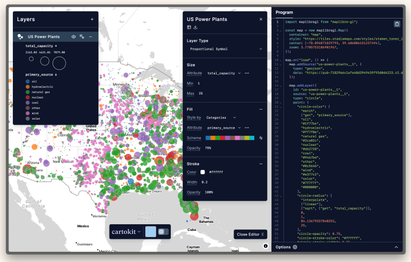
THE USER QUESTION

We had 18 data journalists and researchers use cartokit DM+NL, cartokit, and GitHub Copilot to program published interactive maps from national newsrooms.

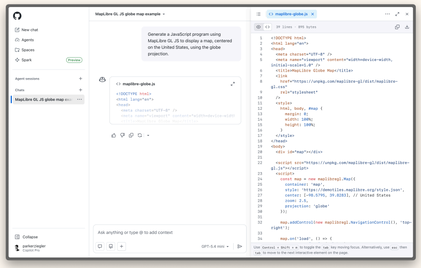
cartokit DM+NL

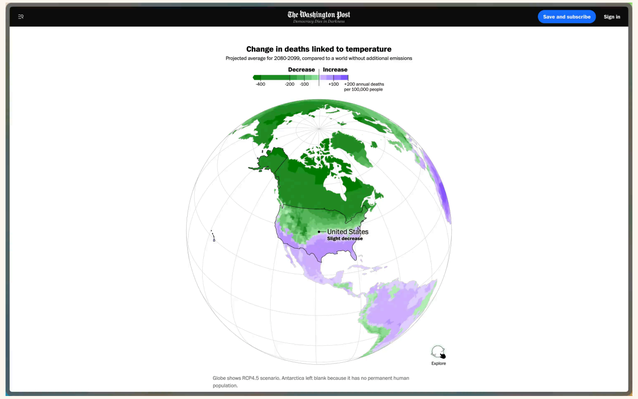


cartokit



GitHub Copilot






```
import { Map, setWorkerUrl } from "maplibre-gl";
import workerUrl from "maplibre-gl/dist/maplibre-gl-worker.mjs?worker&url";

import "maplibre-gl/dist/maplibre-gl.css";
import "./style.css";

setWorkerUrl(workerUrl);

const map = new Map({
  container: "map",
  style: "https://tiles.stadiamaps.com/styles/stamen_toner_lite.json",
  center: [-98.35, 39.5],
  zoom: 4,
});

map.on("style.load", () => {
  map.setProjection({ type: "globe" });
});

map.on("load", () => {
  map.addSource("climate-impact-regions__2", {
    type: "geojson",
    data: "https://pub-...r2.dev/wapo-climate-impact-regions.json",
  });

  map.addLayer({
    id: "climate-impact-regions__1",
    source: "climate-impact-regions__1",
    type: "fill",
    paint: {
      "fill-color": [
        "step",
        ["get", "years_2020_2039"],
...
```

JS

PART 1

Technical Approach

PART 2

User Study Findings

PART 3

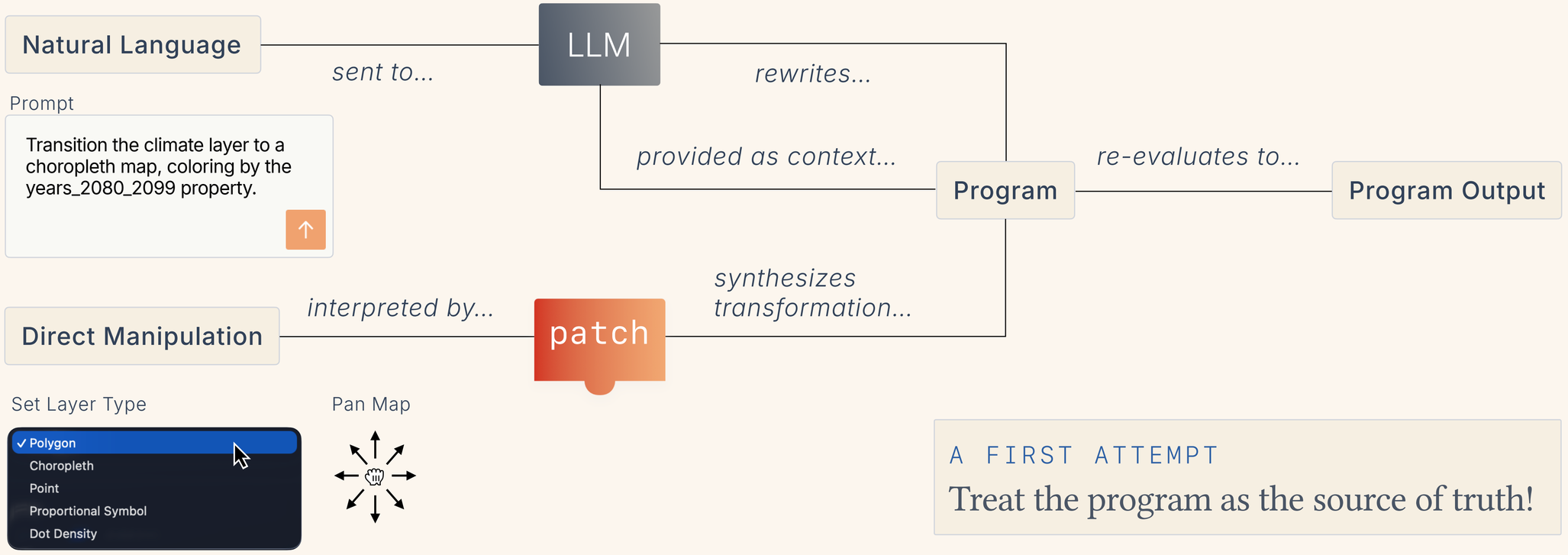
Looking Ahead

PART 1

Technical Approach

KEY CHALLENGE

How do we support **both direct manipulation and natural language editing** at **interactive speed** while **ensuring correctness** of the generated program *and* program output?



A FIRST ATTEMPT

Treat the program as the source of truth!

ISSUE 1

LLM cannot **guarantee the semantic correctness** of generated code

Natural Language

sent to...

LLM

rewrites...

Prompt

Transition the climate layer to a choropleth map, coloring by the years_2080_2099 property.

provided as context...

Program

re-evaluates to...

Program Output

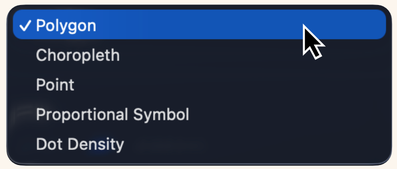
Direct Manipulation

interpreted by...

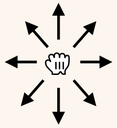
patch

synthesizes transformation...

Set Layer Type



Pan Map



A FIRST ATTEMPT

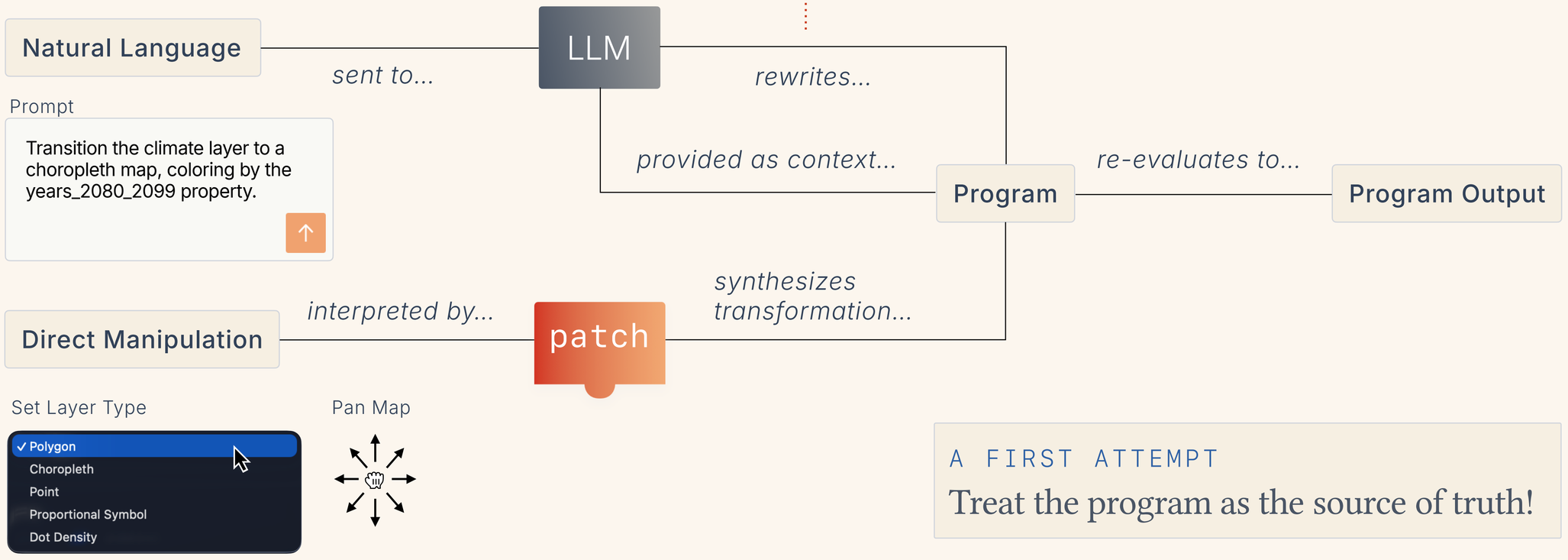
Treat the program as the source of truth!

ISSUE 1

LLM cannot **guarantee the semantic correctness** of generated code

ISSUE 2

“Progressive degradation” of DM editing



A FIRST ATTEMPT

Treat the program as the source of truth!

ISSUE 1

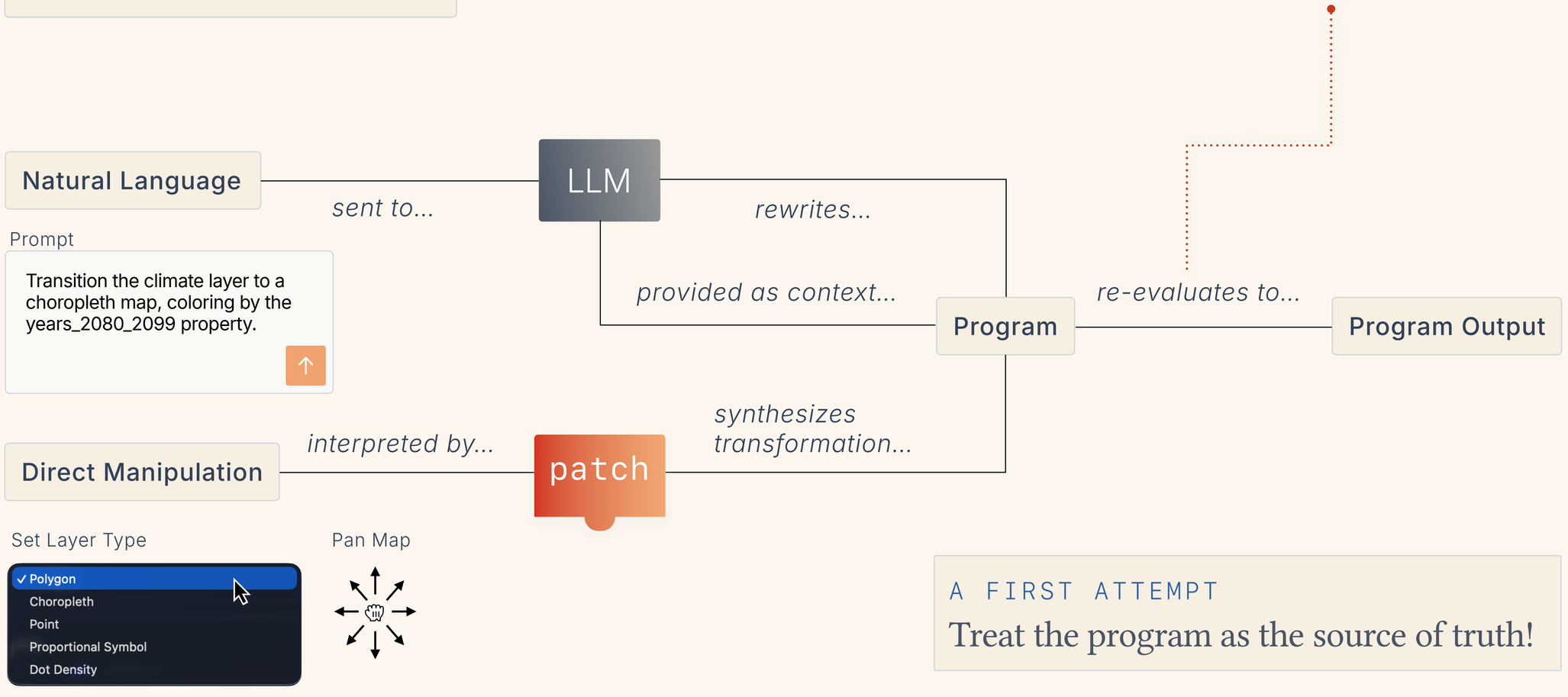
LLM cannot **guarantee the semantic correctness** of generated code

ISSUE 2

"Progressive degradation" of DM editing

ISSUE 3

Every edit requires **full program re-evaluation**



A FIRST ATTEMPT

Treat the program as the source of truth!

PLDI '25



Fast Direct Manipulation Programming with Patch-Reconciliation Correspondence

PARKER ZIEGLER, University of California, Berkeley, USA
JUSTIN LUBIN, University of California, Berkeley, USA
SARAH E. CHASINS, University of California, Berkeley, USA

Direct manipulation programming gives users a way to write programs without directly writing code, by using the familiar GUI-style interactions they know from direct manipulation interfaces. To date, direct manipulation programming systems have relied on two core components: (1) a *patch* component, which modifies the program based on a GUI interaction, and (2) a *forward evaluator*, which executes the modified program to produce an updated program output. This architecture has worked for developing short-running programs—i.e., programs that reliably execute in <1 second—generating outputs such as SVG and HTML documents. However, direct manipulation programming has not yet been applied to long-running programs (e.g., data visualization, mapping), perhaps because executing such programs in response to every GUI interaction would mean crossing outside of interactive speeds. We propose extending direct manipulation programming to long-running programs by pairing a standard *patch* component (**patch**) with a corresponding *reconciliation* component (**recon**). **recon** directly updates the program *output* in response to a GUI interaction, obviating the need for forward evaluation.

We introduce corresponding **patch** and **recon** procedures for the domain of geospatial data visualization and prove them sound—that is, we show that the output produced by **recon** is identical to the output produced by forward-evaluating a **patch**-modified program. **recon** can operate both incrementally and in parallel with **patch**. Our implementation of our **patch-recon** instantiation achieves a 2.92× median reduction in interface latency compared to forward evaluation on a suite of real-world geospatial visualization tasks. Looking forward, our results suggest that *patch-reconciliation correspondence* offers a promising pathway for extending direct manipulation programming to domains involving large-scale computation.

CCS Concepts: • **Human-centered computing** → **User interface programming**; • **Software and its engineering** → **Graphical user interface languages**; **Integrated and visual development environments**.

Additional Key Words and Phrases: direct manipulation, direct manipulation programming, reconciliation, patch-reconciliation correspondence, cartokit, geospatial data

ACM Reference Format:
Parker Ziegler, Justin Lubin, and Sarah E. Chasins. 2025. Fast Direct Manipulation Programming with Patch-Reconciliation Correspondence. *Proc. ACM Program. Lang.* 9, PLDI, Article 175 (June 2025), 26 pages. https://doi.org/10.1145/3729278

ISSUE 3

Every edit requires **full program re-evaluation**

SOLUTION

Use **patch-recon** with an **edit language** to **avoid re-evaluation entirely**

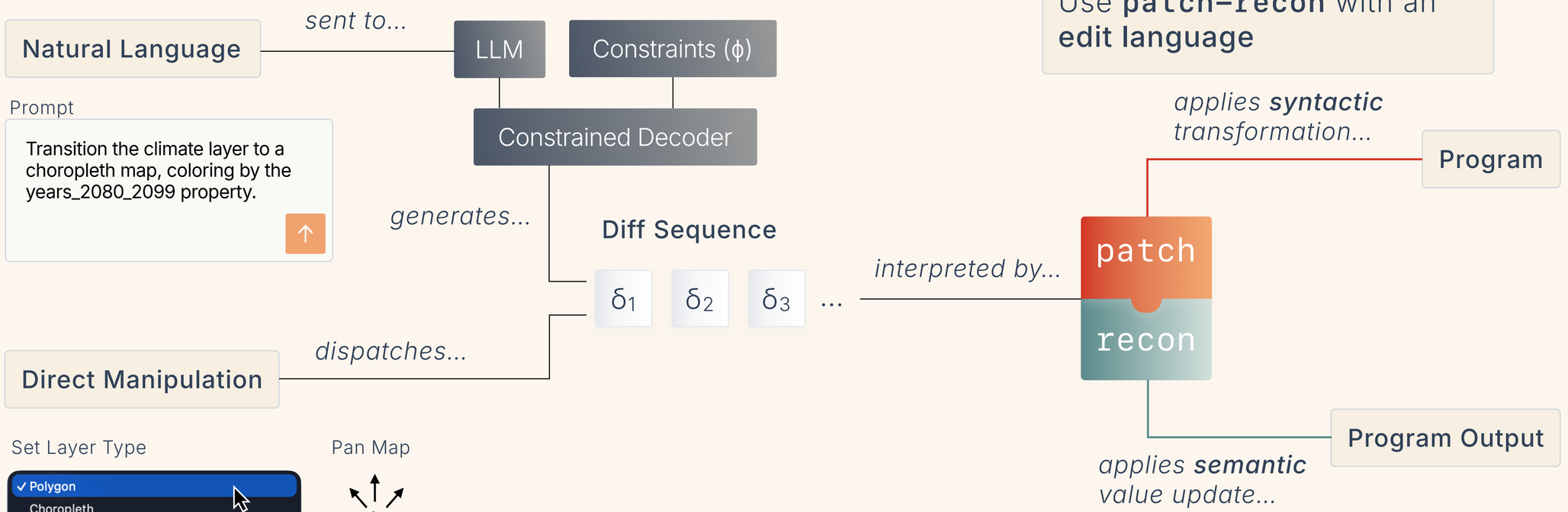
Diff δ ::= setChannel(id, C, fn)
| removeChannel(id, C)
| addLayer(id, ℓ)
| removeLayer(id)
| transformLayer(id, transform)
| setCamera(Pr, Z, [Lng, Lat])

ISSUE 3

Every edit requires **full program re-evaluation**

SOLUTION

Use **patch–recon** with an **edit language**



ISSUE 1

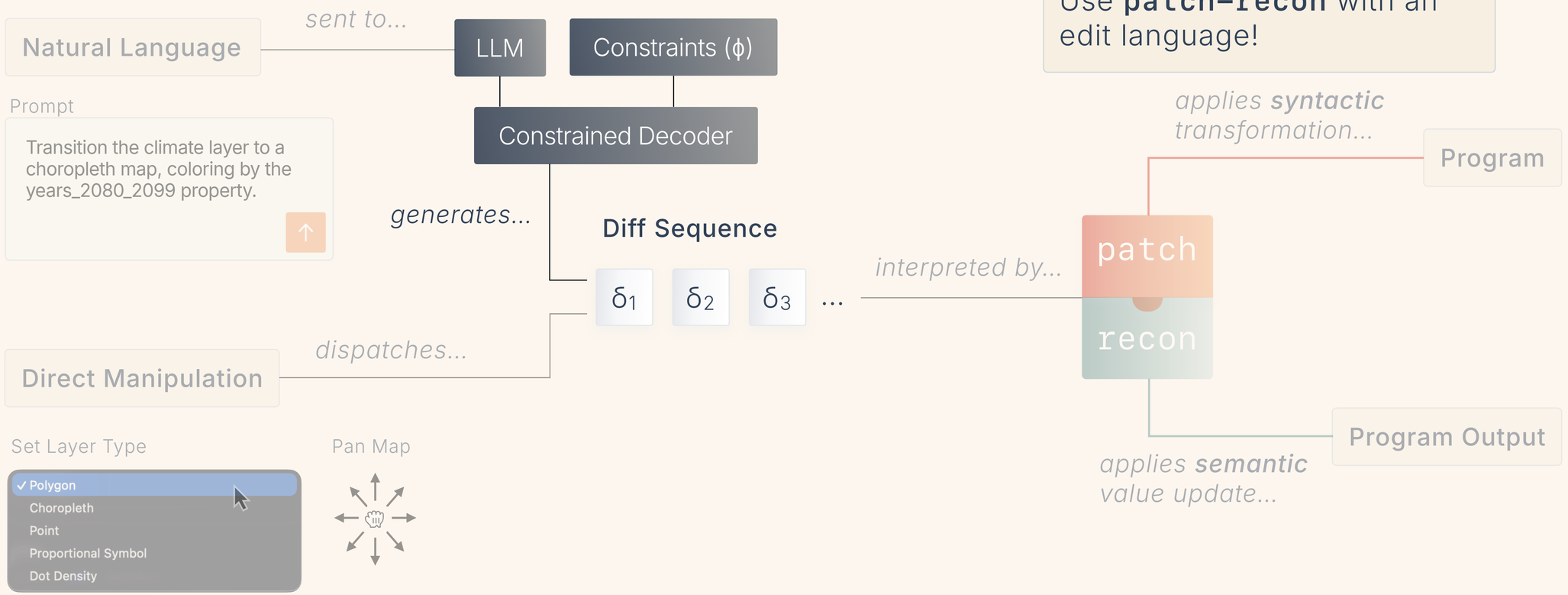
LLM cannot **guarantee the semantic correctness** of generated code

ISSUE 3

Every edit requires **full program re-evaluation**

SOLUTION

Use **patch–recon** with an edit language!



ISSUE 1

LLM cannot **guarantee the semantic correctness** of generated code

ISSUE 2

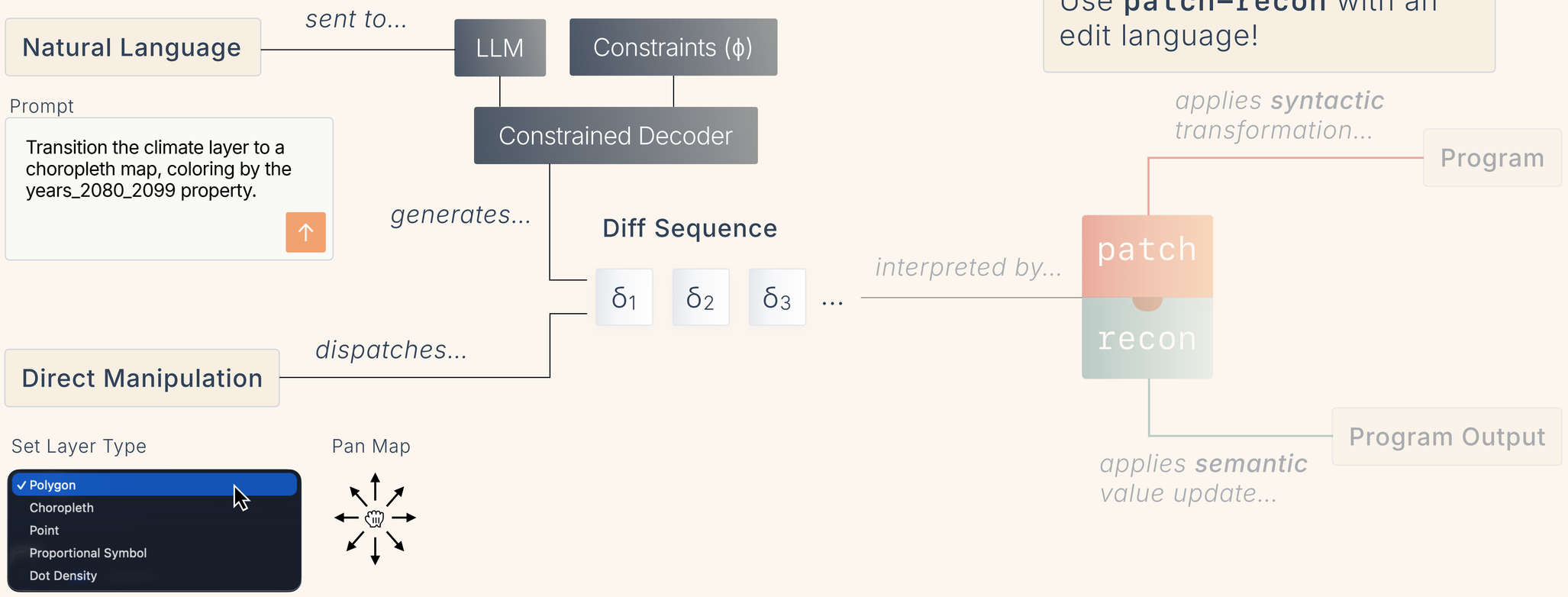
"Progressive degradation" of DM editing

ISSUE 3

Every edit requires **full program re-evaluation**

SOLUTION

Use **`patch-recon`** with an edit language!



KEY CHALLENGE

How do we support **both direct manipulation and natural language editing** at **interactive speed** while **ensuring correctness** of the generated program *and* program output?

KEY TAKEAWAY

By modeling programs as **sequences of structured edits (δ)** in an **edit language** $\mathcal{L}$:

$$e = (\delta_n \circ \ldots \circ \delta_1)\, e_0$$

we simplify the task of both modalities to **constrained edit sequence generation**.

PART 1

Technical Approach

PART 1

Technical Approach

PART 2

User Study Findings

PART 3

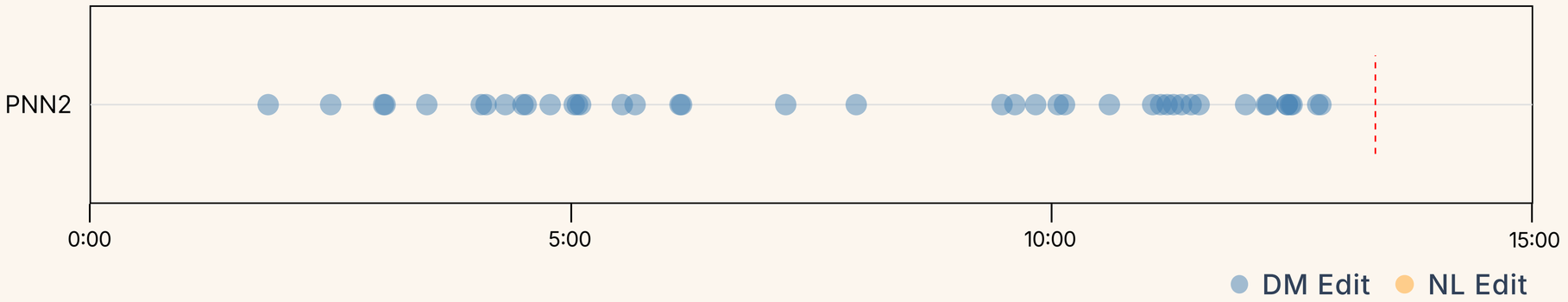
Looking Ahead

PART 2

User Study Findings

Of the 2,946 edits participants made, roughly **96%** of them were made **via direct manipulation**. Just 181 edits (**4%**) were made via **natural language prompting**.

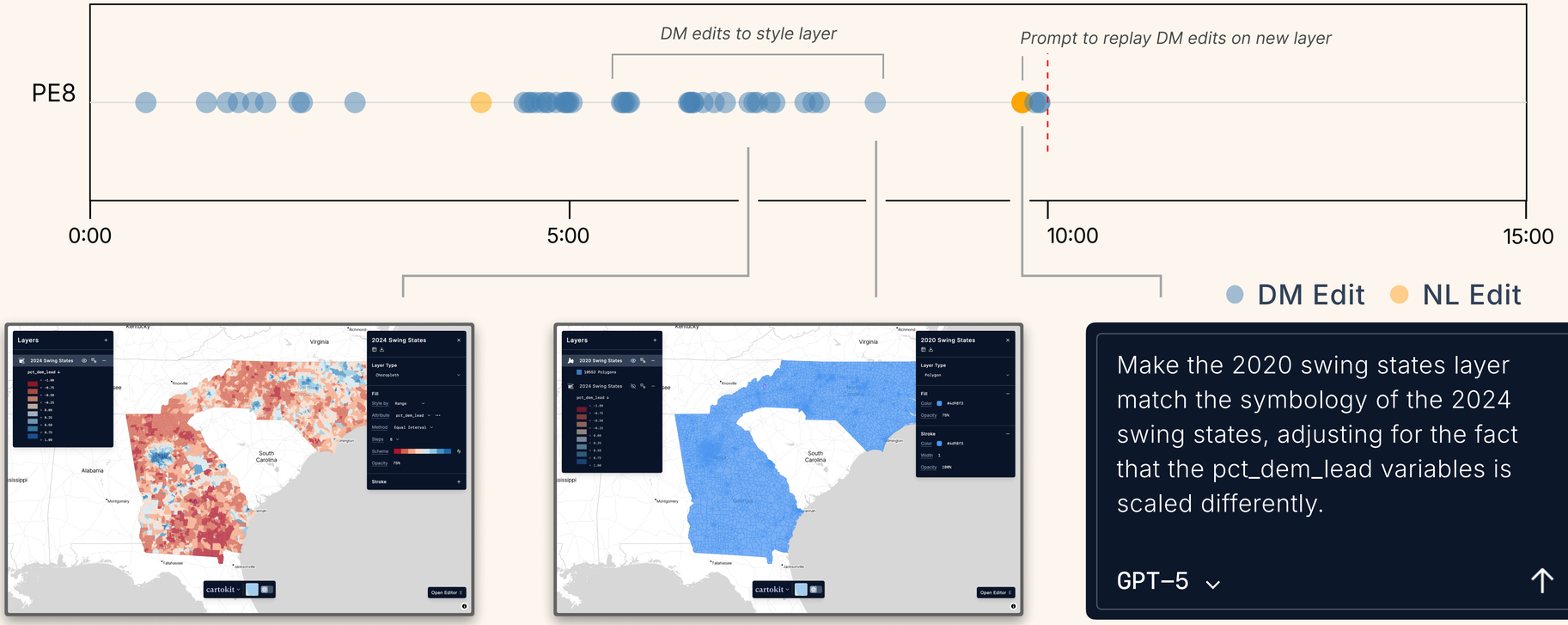
DM Only. PNN2 did not issue a single prompt in the course of their Reproduction task.



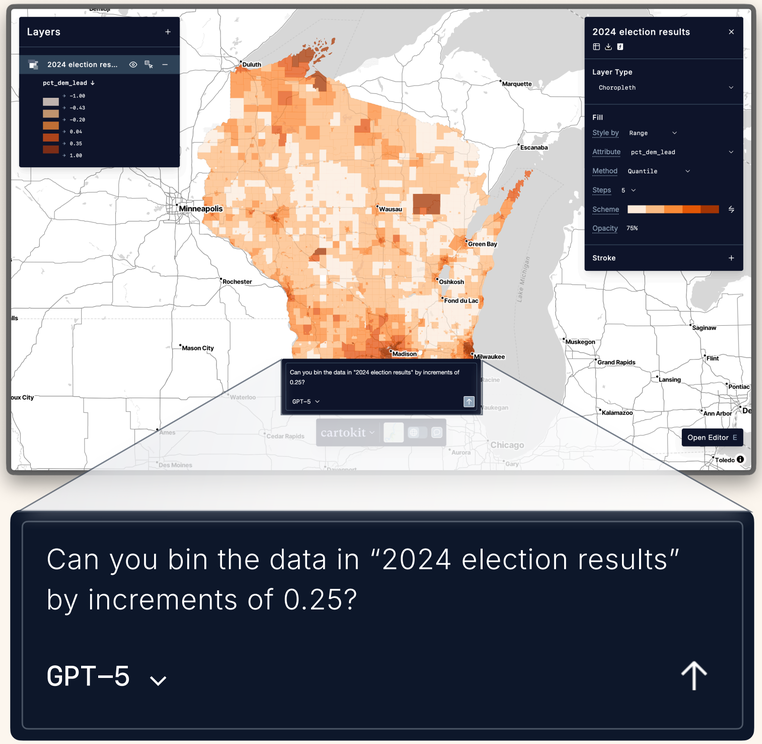
> “I definitely did forget that [the LLM] was a feature, but also, looking back, I don't know what I really would've needed it for because I was directly manipulating the map and I didn't need help to figure that out … I thought I could just accomplish everything without having to declare bankruptcy and be like, ‘You [LLM] tell me how to do this, I don't really know.’”
>
> — PE7

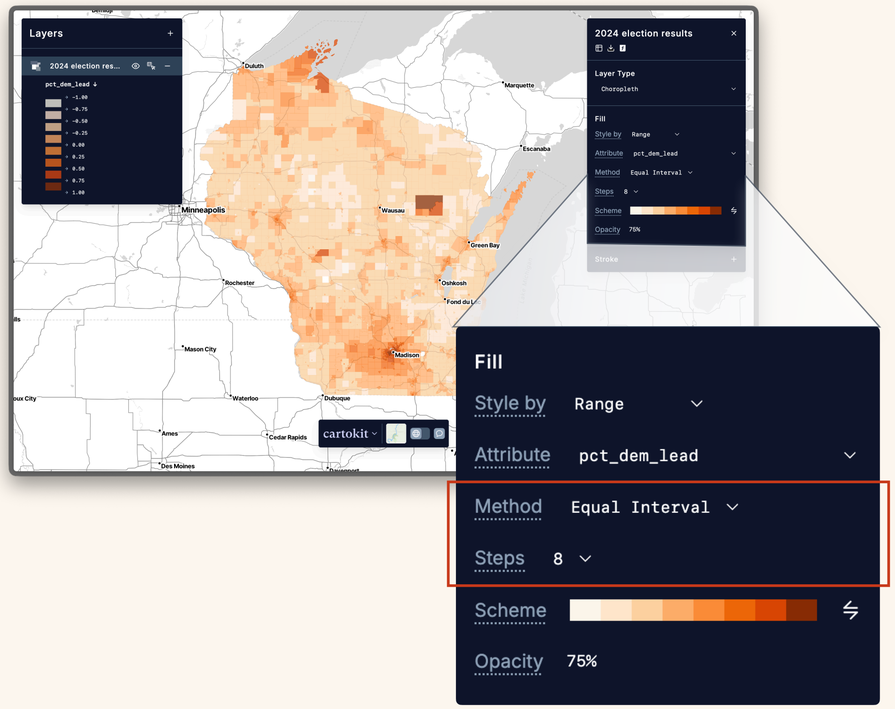
When participants did use natural language, it was primarily to **automate, parameterize, and replay repetitive edits**.

DM → NL: Abstracting and replaying edits. PE8 performed a large sequence of direct manipulation edits, then prompted to have the LLM apply the same sequence of edits to a different layer.

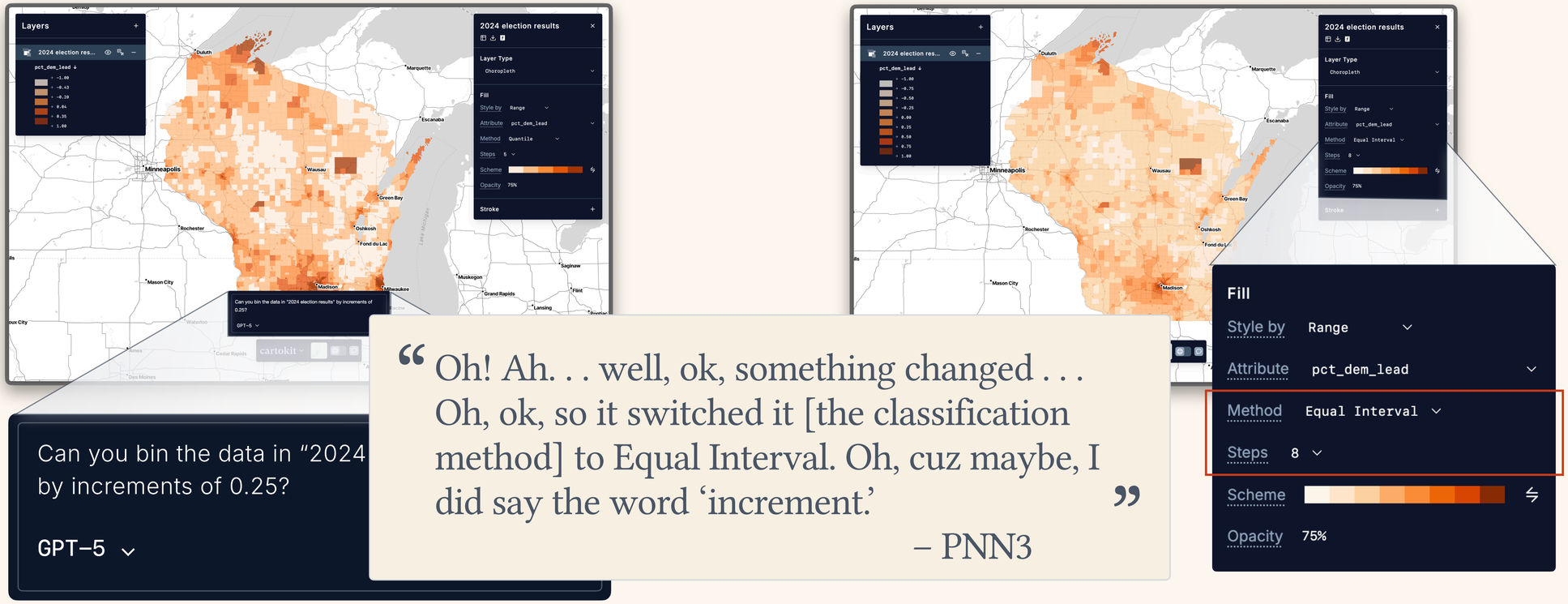


Participants used **the state of GUI elements** as “gauges” that helped them **interpret the scope of LLM-triggered edits**.



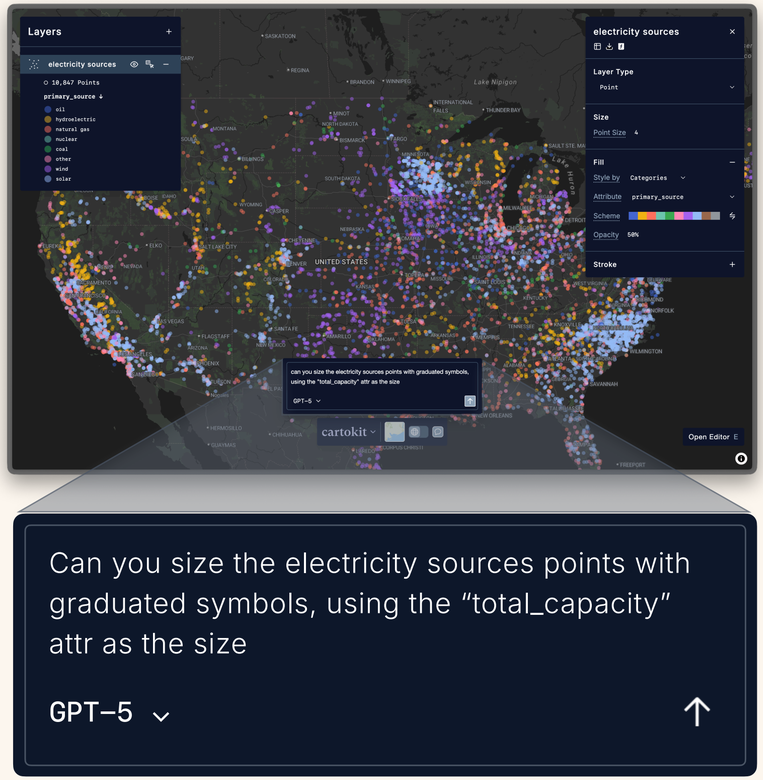


Participants used **the state of GUI elements** as “gauges” that helped them **interpret the scope of LLM-triggered edits**.



> “Oh! Ah. . . well, ok, something changed . . . Oh, ok, so it switched it [the classification method] to Equal Interval. Oh, cuz maybe, I did say the word ‘increment.’”
>
> – PNN3

The GUI both **“revealed” the edit language** and **helped participants internalize the kind of programs it can express**, influencing how they wrote prompts.

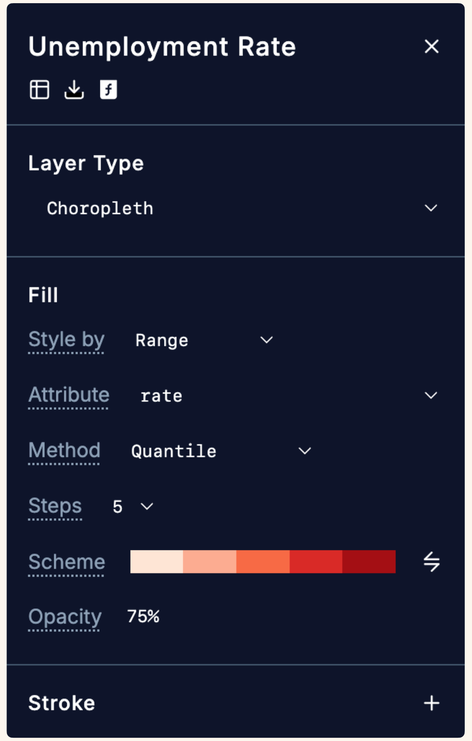


> “Knowing what the tool was capable of before I started prompting was important. I wasn’t doing joins and different spatial analysis things because I didn’t see those tools in [the GUI]. I was constrained by what I knew I could do without the chatbot.”
>
> – PNN3

This **conceptual scaffolding** also helped them **write better prompts** even when direct manipulation was not available.

PE2's prompt to **GitHub Copilot** **after** using **cartokit**

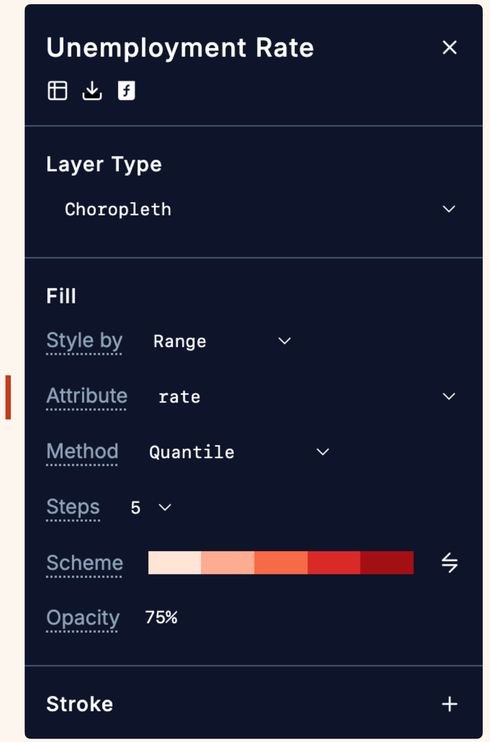
> "Make a **choropleth map** using the following US unemployment data. Use the **rate property** to determine the darkness of the polygon color. **Use red. Darker colors should correspond with higher numbers. Bin the colors along quintiles.** *<Data URL>*"



This **conceptual scaffolding** also helped them **write better prompts** even when direct manipulation was not available.

PE2's prompt to **GitHub Copilot** **after** using **cartokit**

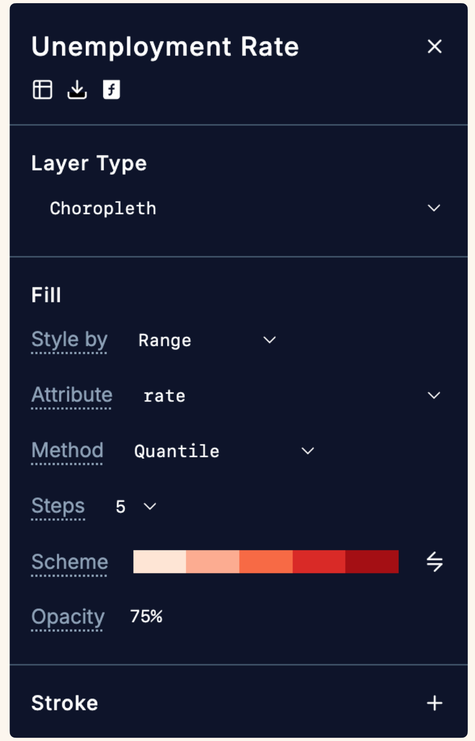
> "Make a **choropleth map** using the following US unemployment data. Use the **rate property** to determine the darkness of the polygon color. **Use red. Darker colors should correspond with higher numbers. Bin the colors along quintiles.** *<Data URL>*"



This **conceptual scaffolding** also helped them **write better prompts** even when direct manipulation was not available.

PE2's prompt to GitHub Copilot **after** using cartokit

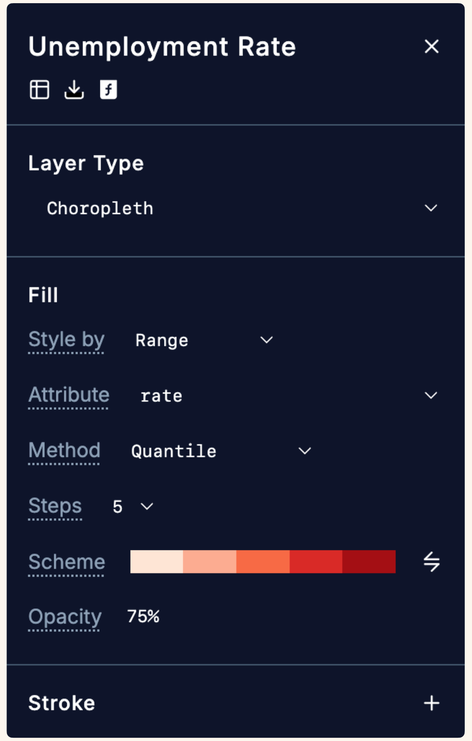
> "Make a **choropleth map** using the following US unemployment data. Use the **rate property** to determine the darkness of the polygon color. **Use red. Darker colors should correspond with higher numbers. Bin the colors along quintiles.** *<Data URL>*"



This **conceptual scaffolding** also helped them **write better prompts** even when direct manipulation was not available.

PE2's prompt to GitHub Copilot **after** using cartokit

> "Make a **choropleth map** using the following US unemployment data. Use the **rate property** to determine the darkness of the polygon color. **Use red. Darker colors should correspond with higher numbers. Bin the colors along quintiles.** *<Data URL>*"



PART 2

User Study Findings

PART 1

Technical Approach

PART 2

User Study Findings

PART 3

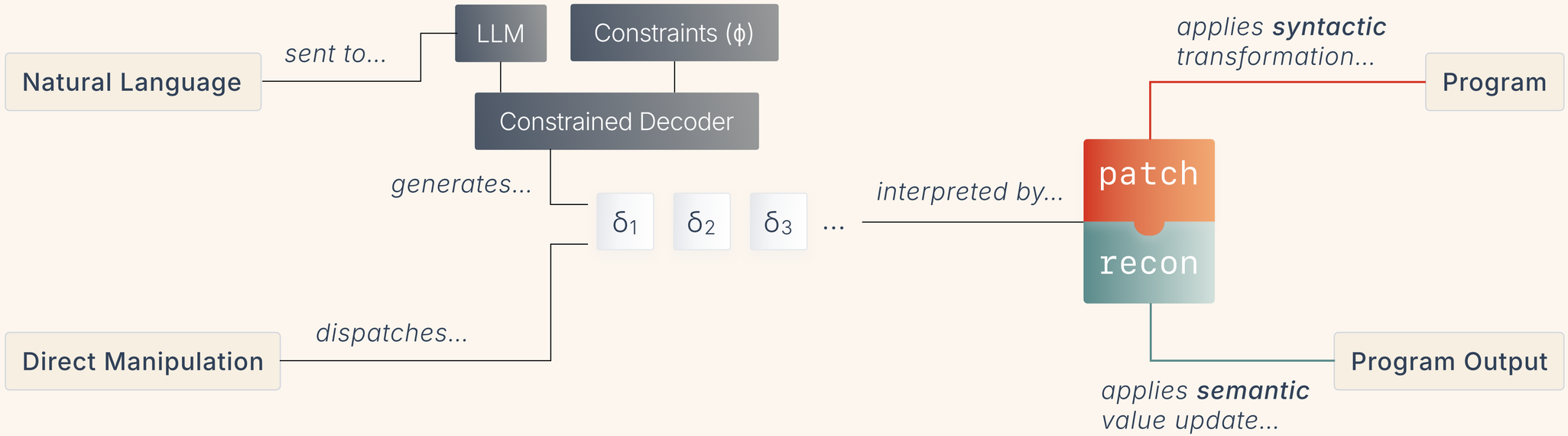
Looking Ahead

PART 3

Looking Ahead

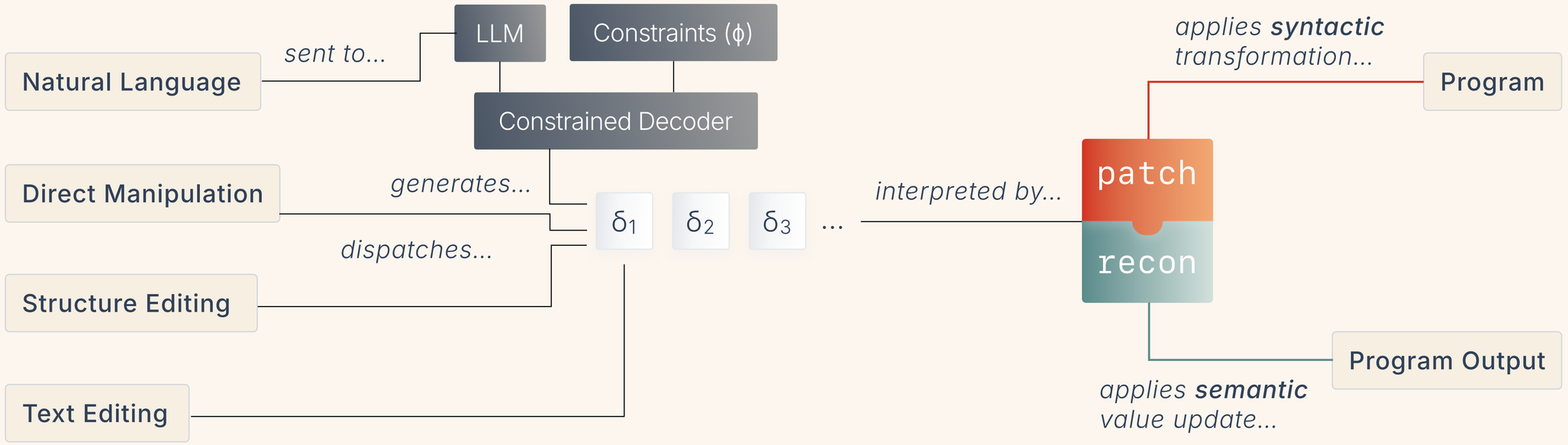
KEY TAKEAWAY

Edit languages can be an extremely powerful interface for combining the the **performance and correctness** benefits of **structured programming techniques** with **natural language** as an editing modality.



KEY TAKEAWAY

Natural language isn't necessarily the one modality developers want! **Pluralistic programming systems** that support **moving between modalities** based on editing context may be a promising pathway for new systems.



Direct Manipulation and NATURAL LANGUAGE Programming Together at Last?

Read the Full Paper

Used by Journalists at

The New York Times · The Washington Post · WSJ · Grist · HCN

Try out cartokit!



https://alpha.cartokit.dev/

https://github.com/parkerziegler/cartokit/



Parker Ziegler
https://parkie-doo.computer/
peziegler@cs.berkeley.edu



David Minh-Duy Cao
https://cao.sh/
dcao@berkeley.edu



Justin Lubin
https://jlubin.net/
justinlubin@berkeley.edu



Sarah E. Chasins
https://schasins.com/
schasins@cs.berkeley.edu

OOPSLA '24
7 Oct 2024