Probabilistic Program Synthesis

From Lazy Categorical Semantics to LLM-Guided Model Synthesis

Younesse Kaddar · Supervised by Sam Staton

Confirmation of Status Viva · *May 1st, 2025*

Why combine PPLs & LLMs?

Probabilistic Prog. Languages

👍
- Explicit causal structure
- Principled uncertainty
- Formal semantics

👎
- Needs expert domain knowledge
- Tedious to write for large systems
- No one-size-fits-all inference

Large Language Models

👍
- Massive pretraining data
- Natural-language priors
- General-purpose CoT reasoning

👎
- Black-box
- Poor uncertainty calibration
- No formal meaning

🔺 **Goal:** Use LLMs to *propose* models, but keep PPLs as the *structured output* for modelling.

Thesis Structure & Key Papers

Part I: Semantics of PPL

1. Categorical Semantics for PPLs

- 📄 "Probabilistic Programming for Random Graphs" (POPL '24)

2. LazyPPL for Bayesian Nonparametrics

- 📄 "Affine Monads and Lazy Structures" (POPL '23)
- 💻 LazyPPL Haskell library

3. Semantics of Stochastic Memoization

- 📄 "Stochastic Memoization via Presheaves" (MFPS '23)

Part II: LLM-Guided PPL Synthesis

4. GFlowNets & LLM Steering

- 📄 "Amortizing inference in LLMs" (ICLR '24)

5. LLM-Guided Probabilistic Program Synthesis

- 🔧 SynthStats with GFlowNet-guided LLMs (in progress)

6. Applications and Case Studies

- 📄 "Can a Bayesian Oracle Prevent Harm?" (2024)

Appendix : Additional LazyPPL Examples, Extended Proofs, Implementation Details

Part I: Foundational Work (LazyPPL Focus)

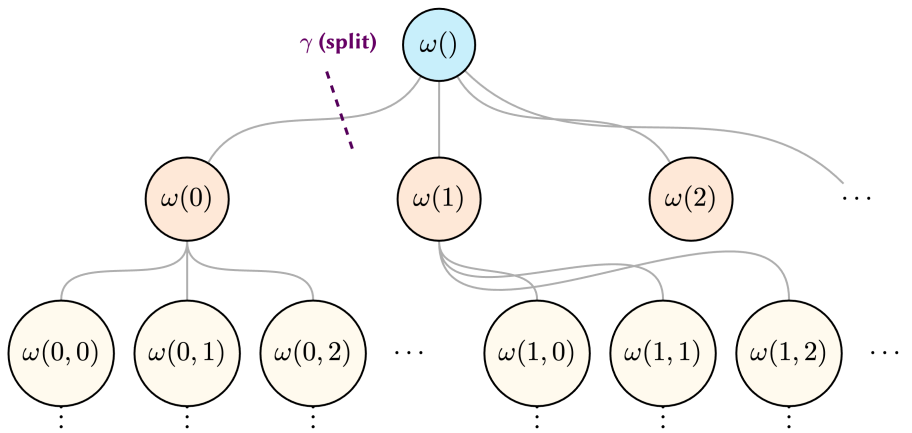
- **LazyPPL** (POPL '23)
  - Haskell library for Bayesian nonparametrics
  - Two commutative monads design:
    - `Prob a` : Affine, sampling
    - `Meas a` : Non-affine, scoring
  - Quasi-Borel spaces semantics
- **Infinite Lazy Rose Tree**: Ω
  - Nodes: $U(0,1)$ values
  - Branches: Independent vars
  - Generated on demand

```
data Tree = Tree Double [Tree]
newtype Prob a = Prob (Tree -> a)
```

- **Stochastic Memoization** (MFPS '23)
  - Preserves *data-flow* property
  - Key for nonparametric models

$$T(X)(g) \triangleq \left(P_{\mathrm{f}} \int^{g \hookrightarrow h} \left(X(h) \times [0,1]^{(h-g)_L} \right) \right)^{[0,1]^{g_L}}$$

g: bigraph (memo state), P_f: finite prob., $\int^{g \hookrightarrow h}$: coend over embeddings, g_L: functions

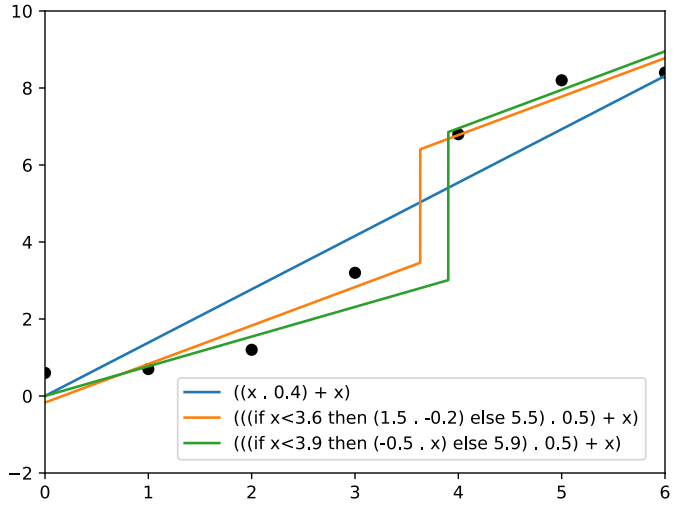


LazyPPL Example: Program Induction

```haskell
-- A tiny language for arithmetic expressions
data Expr = Var | Constt Double | Add Expr Expr
          | Mult Expr Expr | IfLess Double Expr Expr

-- Generate random expressions
randexpr :: Prob Expr
randexpr = do
  i <- categorical [0.3,0.3,0.18,0.18,0.04]
  es <- sequence [...]  -- PCFG
  return $ es !! i

-- Interpret an expression
eval :: Expr -> Double -> Double

-- Regression over programs
regress :: [(Double, Double)] -> Meas Expr
regress dataset = do
  expr <- sample randexpr  -- Sample from prior
  -- Score by likelihood of data
  forM_ dataset $ \(x, y) ->
    score $ normalPdf (eval expr x) 0.1 y
  return expr
```

> **Task**: Infer programs from input-output examples

- **Compositional semantics** enables:
  - Declarative style
  - Infinite-dimensional spaces
  - No arbitrary truncation



Inferred programs fitting the data

Part II – SynthStats Pipeline

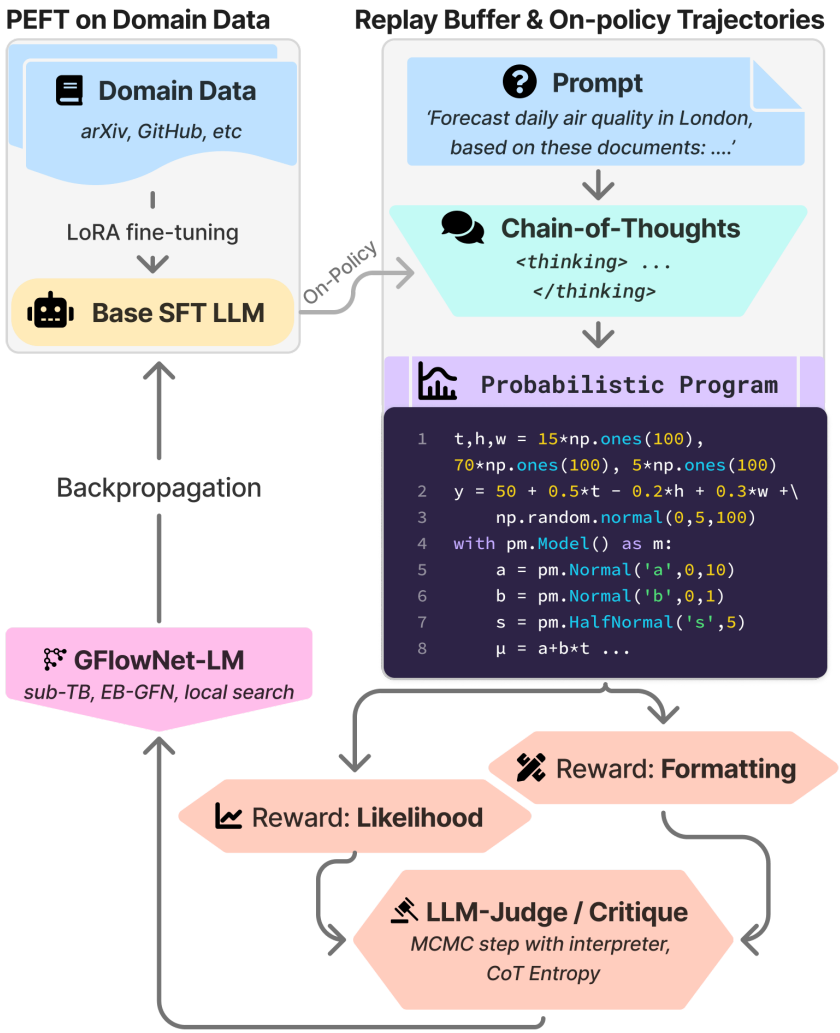
GFlowNet-Fine-tuned LLM (ICLR '24)

- Learns a *distribution* $P(\text{program}) \propto R$ (vs RL maximizing for single best one)
- Generates:
  1. Chain-of-thought reasoning
  2. Then: Executable PPL code (PyMC)

📄 Based on: *"Amortizing intractable inference in LLMs"* (ICLR '24)

Key Benefits

- Diverse models → better uncertainty
- More robust to *reward hacking*
- Interpretable, auditable code



Roadmap & Take-aways

Timeline to Submission (Michaelmas 2025)

Trinity 2025	Summer 2025
🔹 SynthStats prototype	🔹 Complete experiments
🔹 Multi-component reward	🔸 Thesis writing
🔸 Thesis writing	📌 *JAX backend (if time permits)*

Key Contributions

- **Bridge**: PPL ⇄ modern LLMs
- **Distribution-based** generation > reward-max RL
- **Interpretable** models vs black-box answers
- Path to *safer, uncertainty-aware* ML

Thank You!

Questions & Feedback Welcome