Théo Matricon

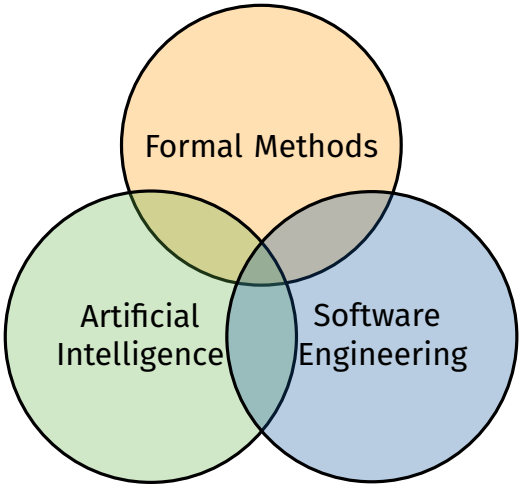
2025-current:

Postdoc in LLM4Code with Mathieu Acher
Code Generation and LLMs
DiverSE, Software Engineering Team
IRISA, Rennes $\rightarrow$ medical condition (getting RQTH)

2021-2024:

PhD supervised by Nathanaël Fijalkow
Scaling domain agnostic techniques for program Synthesis
M2F, Formal Methods Team
LaBRI, Bordeaux

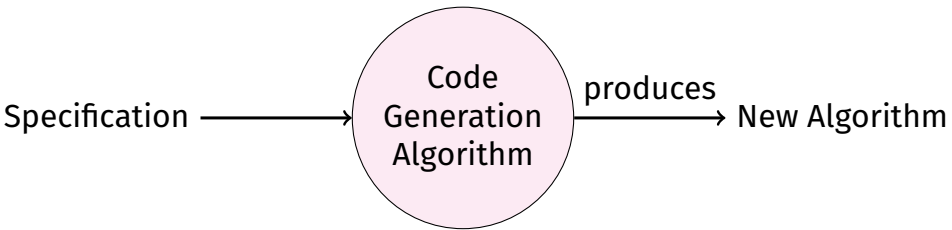
My Research Contributions so far



Selected Publications:

- 2025, submitted to CAV
- 2025, AAAI (+oral: top 20%)
- 2025, IST
- 2022, AAAI (+oral: top 20%)
- 2021, CP

Motivation


Specification
Code Generation Algorithm
produces
New Algorithm

In practice

Specifications:

Logic:
$\forall a, b$
$f(a, b) \geq a$
$f(a, b) \geq b$
$f(a, b) \in \{a, b\}$

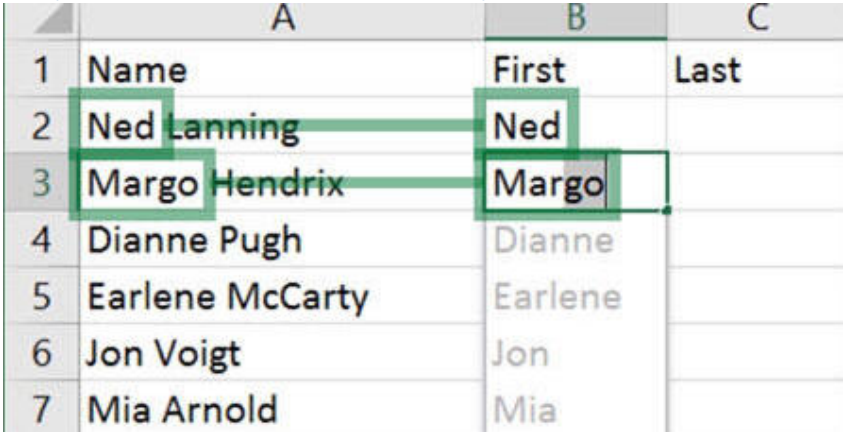
Examples:
$f(1, 5) = 5$
$f(2, 1) = 2$
$f(-3, -9) = -3$

Natural language:
'Write a function that takes the maximum of its two arguments.'

Produced Algorithms:

```
def max(a: int, b: int) ->int:
    if a <=b:
        return b
    else:
        return a
```

Program Synthesis: From Examples



	A	B	C
1	Name	First	Last
2	Ned Lanning	Ned	
3	Margo Hendrix	Margo	
4	Dianne Pugh	Dianne	
5	Earlene McCarty	Earlene	
6	Jon Voigt	Jon	
7	Mia Arnold	Mia	

Copyright Microsoft for syntactic manipulation,
based on papers by *Gulwani* and extensions by ***Matricon*** *et al.*

Program Synthesis: Problem

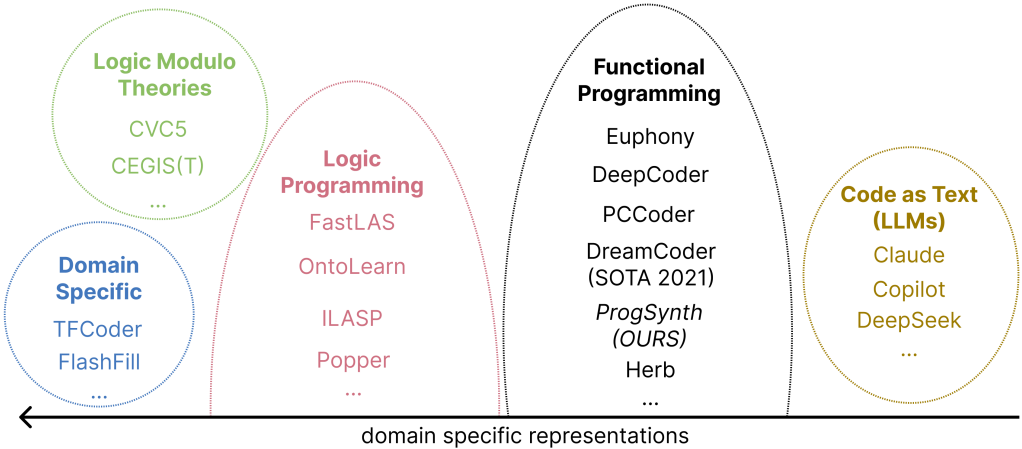
Input:

- the search space G:
  a deterministic tree grammar
- a specification $\mathcal{C}$:
  it checks if a program $p \in \mathcal{L}(G)$ matches the specification

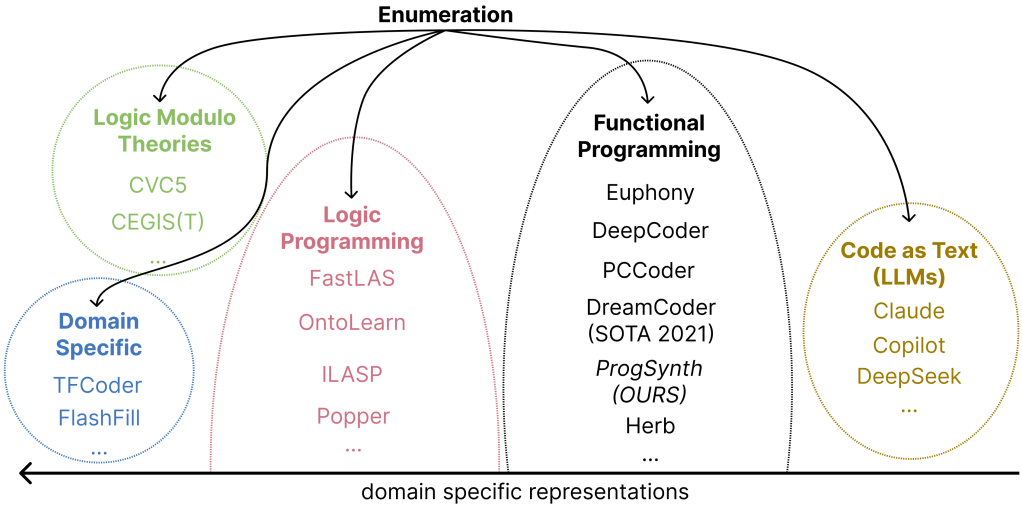
Output:

- a program in the search space that matches the specification:
  a $p \in \mathcal{L}(G)$ such that $\mathcal{C}(p) = \checkmark$

Frameworks


Logic Modulo Theories
CVC5
CEGIS(T)
...
Domain Specific
TFCoder
FlashFill
...
Logic Programming
FastLAS
OntoLearn
ILASP
Popper
...
Functional Programming
Euphony
DeepCoder
PCCoder
DreamCoder (SOTA 2021)
ProgSynth (OURS)
Herb
...
Code as Text (LLMs)
Claude
Copilot
DeepSeek
...
domain specific representations

Frameworks


Enumeration
Logic Modulo Theories
CVC5
CEGIS(T)
...
Logic Programming
FastLAS
OntoLearn
ILASP
Popper
...
Functional Programming
Euphony
DeepCoder
PCCoder
DreamCoder (SOTA 2021)
ProgSynth (OURS)
Herb
...
Code as Text (LLMs)
Claude
Copilot
DeepSeek
...
Domain Specific
TFCoder
FlashFill
...
domain specific representations

Enumeration Problem

Input:

- the search space G:
  a deterministic tree grammar with a cost for each tree
- a specification C:
  it checks if a program $p \in \mathcal{L}(G)$ matches the specification

Goal:

Enumerate all programs in order of non-decreasing costs

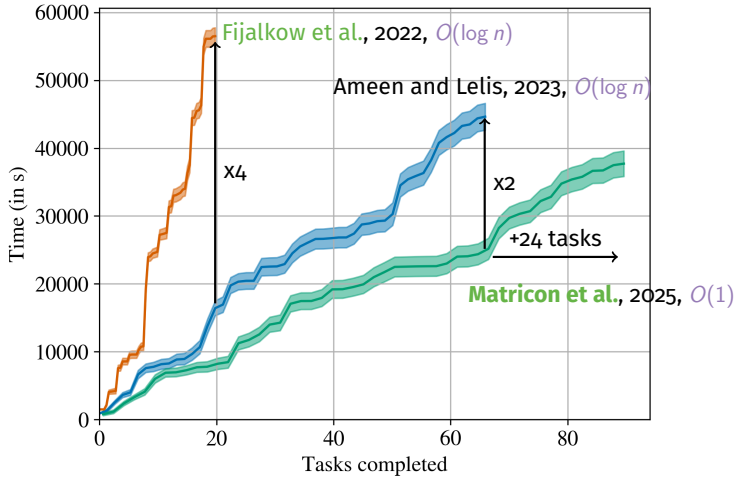
Delay:

Time complexity in terms of n: number of programs enumerated
Between enumeration of the n^{th} program and the next

Overview

Major papers:

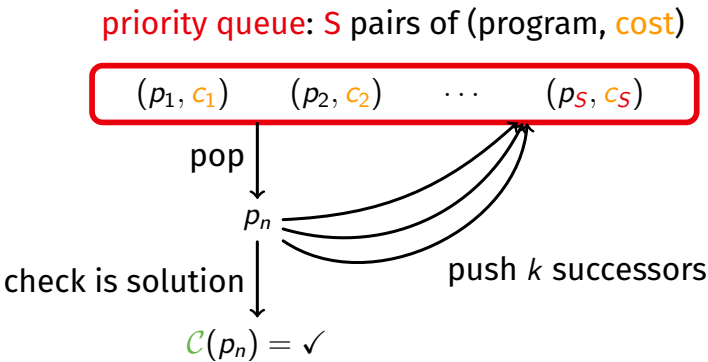
- 2017, machine learning + enumeration, *Balog et al.*, ICLR
- 2018, $O(\log n)$, *Lee et al.*, PLDI



Skeleton of an enumeration algorithm

At step n:

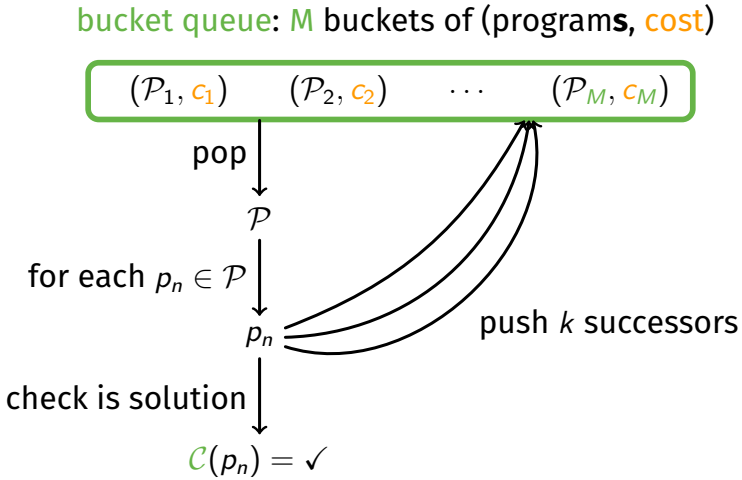
$S = O(n)$
$k = O(1)$



Our enumeration algorithm

At step n:

$M = O(1)$
$k = O(1)$



Our contribution

We prove bounded differences in cost:
$\exists M,\ \forall n,\ cost\_next(p_n) - cost(p_n) \leq M$

This implies: priority queues $O(\log n)$ $\rightarrow$ bucket queues $O(1)$.

Impact

Published in **AAAI 2025** (+oral: 20% of accepted papers).
Fastest ranked enumeration for program synthesis in practice.
First algorithm with $O(1)$ delay $\rightarrow$ closes open question.

My Research Project

Observations:

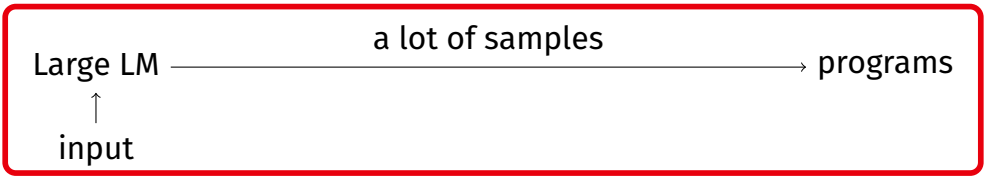
- LLMs are the new state of the art for code generation
- Resource-heavy, expensive and slow
- Exponential increase in data/parameters $\rightarrow$ linear increase in performance
- Unreliable $\rightarrow$ only use them as a direction for search

Scaling Code Generation to be reliable and better!

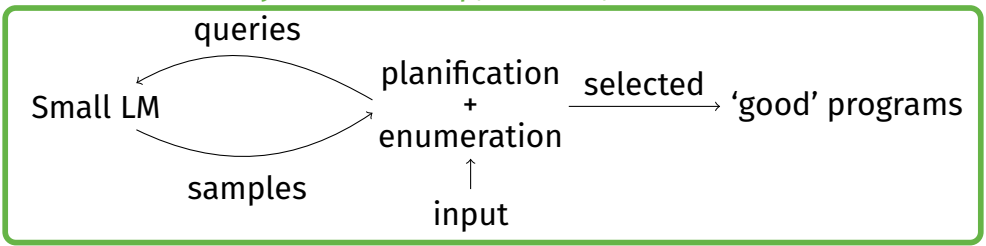
Different code generation paradigm!
Reliable by design, better code with faster generation.

My Research Project

Current Approaches (costly, unreliable, slow)



My Vision (cheap, reliable, fast)



Axis 1: Hierarchical Code Generation

```
class DatabaseConnector:
    def open_connection(self, ip: str) ->None:
        do_stuff(self, ip)

    def close_connection(self) ->None:
        if not some_condition():
            raise SomeError()
        do_other_stuff(self)

    def query(self, query: str) ->str:
        return do_thing(self, query)
```

Axis 1: Hierarchical Code Generation

Structure (Easy?)

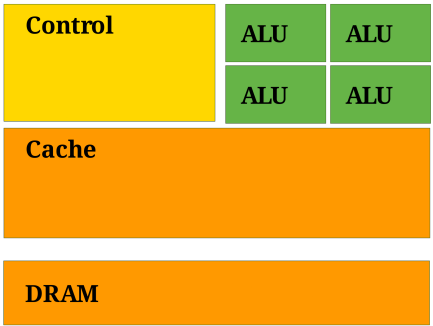
Filling (Hard?)

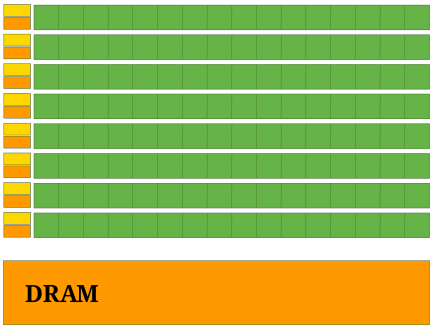
```
class DatabaseConnector:
    def open_connection(self, ip: str) ->None:
        ???

    def close_connection(self) ->None:
        ???

    def query(self, query: str) ->str:
        ???
```

Axis 2: Local GPU-friendly Search





GPUs: massively **parallel** $\rightarrow$ huge speed-up
But **different paradigm**

Axis 2: Local GPU-friendly Search

Current approaches cannot be adapted efficiently.

We need a **new enumeration paradigm** for GPUs.

It requires a **new theoretical** understanding.

Synergies

Long Term Objective:

Axis 1: provides practical reduction of the complexity of the problem.
Axis 2: improves the search efficiency.

Orthogonal directions that can be coupled together for better results!

Integration

LLM4Code project
Cross-Team interactions with FM and AI

→ CRIStAL, UMR 9189, Lille, **Spirals**

SE and AI: Clément Quinton, Romain Rouvoy
Enumeration: Pierre Bourhis (+ **LINKS**)

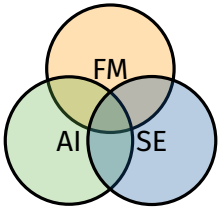
→ IRISA, UMR 6074, Rennes, **DiverSE**

SE and AI: Mathieu Acher, Olivier Barais, Benoît Combemale, Aymeric Blot, Quentin Perez, Djamel E. Khelladi
CodeCommons project

→ LaBRI, UMR 5800, Bordeaux, **Progress**

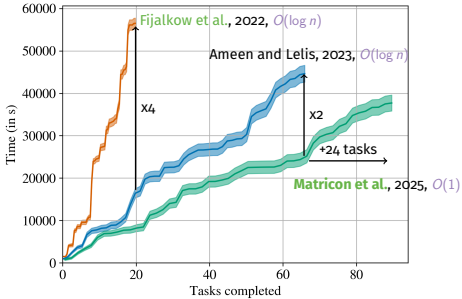
SE and AI: Romain Robbes, Xavier Blanc, Jean-Rémy Falleri and Thomas Degueule
Enumeration: (+ **M2F**)

Théo Matricon



Selected Publications:

- AI: 2 *A** →
- FM: 1 *A*, *A** submitted
- SE: 1 *A*



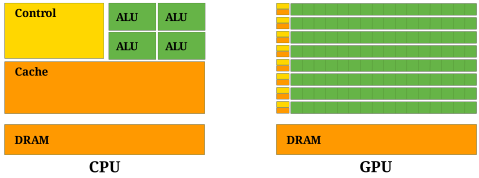
Research Project:

Structure (Easy?) Filling (Hard?)

```python
class DatabaseConnector:
    def open_connection(self, ip: str) ->None:
        ???

    def close_connection(self) ->None:
        ???

    def query(self, query: str) ->str:
        ???
```



[1] G. Bathie, N. Fijalkow, T. Matricon, B. Mouillon, and P. Vandenhove, "LTL$_f$ learning meets boolean synthesis," *under review at CAV*, 2025. [Online]. Available: https://theomat.github.io/files/ltl.pdf.

[2] A. Martin, D. E. Khelladi, T. Matricon, and M. Acher, "Re-evaluating metamorphic testing of chess engines: A replication study," *Information and Software Technology (preprint)*, 2025. [Online]. Available: https://theomat.github.io/files/chess.pdf.

[3] T. Matricon, N. Fijalkow, and G. Lagarde, "Ecosearch: A constant-delay best-first search algorithm for program synthesis," in *International Conference on Artificial Intelligence, AAAI (preprint)*, 2025. [Online]. Available: https://arxiv.org/abs/2412.17330.

[4] G. Shabadi, N. Fijalkow, and T. Matricon, "Theoretical foundations for programmatic reinforcement learning," in *GenPlanAI workshop AAAI (preprint)*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2402.11650.

[5] S. Ameen and L. H. Lelis, "Program synthesis with best-first bottom-up search," *Journal of Artificial Intelligence Research*, vol. 77, pp. 1275–1310, 2023.

[6] T. Matricon, N. Fijalkow, and G. Margueritte, "Wikicoder: Learning to write knowledge-powered code," in *Model Checking Software*, G. Caltais and C. Schilling, Eds., Springer Nature Switzerland, 2023, pp. 123–140, ISBN: 978-3-031-32157-3. [Online]. Available: https://rdcu.be/dITGA.

[7] M. Anastacio, T. Matricon, and H. Hoos, "Challenges of acquiring compositional inductive biases via meta-learning," in *ECMLPKDD Workshop on Meta-Knowledge Transfer*, P. Brazdil, J. N. van Rijn, H. Gouk, and F. Mohr, Eds., ser. Proceedings of Machine Learning Research, vol. 191, PMLR, Sep. 2022, pp. 11–23. [Online]. Available: https://proceedings.mlr.press/v191/anastacio22a.html.

[8] N. Fijalkow, G. Lagarde, T. Matricon, K. Ellis, P. Ohlmann, and A. Potta, "Scaling neural program synthesis with distribution-based search," in *International Conference on Artificial Intelligence, AAAI*, 2022. [Online]. Available: https://arxiv.org/abs/2110.12485.

[9] T. Matricon, N. Fijalkow, G. Lagarde, and K. Ellis, "Deepsynth: Scaling neural program synthesis with distribution-based search," *Journal of Open Source Software*, vol. 7, no. 78, p. 4151, 2022. DOI: 10.21105/joss.04151.

[10] T. Matricon, M. Anastacio, N. Fijalkow, L. Simon, and H. H. Hoos, "Statistical Comparison of Algorithm Performance Through Instance Selection," in *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*, L. D. Michel, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 210, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 43:1–43:21, ISBN: 978-3-95977-211-2. DOI: 10.4230/LIPIcs.CP.2021.43. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CP.2021.43.

[11] W. Lee, K. Heo, R. Alur, and M. Naik, "Accelerating search-based program synthesis using learned probabilistic models," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018, Philadelphia, PA, USA: Association for Computing Machinery, 2018, 436âĂŞ449, ISBN: 9781450356985. DOI: 10.1145/3192366.3192410. [Online]. Available: https://doi.org/10.1145/3192366.3192410.

[12] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "Deepcoder: Learning to write programs," *arXiv preprint arXiv:1611.01989*, 2016.

[13] S. Gulwani, "Automating string processing in spreadsheets using input-output examples," in *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, 2011. [Online]. Available: https://doi.org/10.1145/1926385.1926423.