LLMに書かせる前提で設計された言語の設計について議論して。

••• 回答を生成しています……

At 関数型まつり 2026　July 12, 2026

びしょ〜じょ

発表者の情報を示して

🏢 株式会社 HERP

📎 Nymphium

❤️ OCaml

🤓 趣味:

- ocaml/ocaml/pulls を巡回
- プログラム言語づくり

こんにちは、
びしょ〜じょです。

イントロを書いて

みなさん……

イントロを書いて

みなさん……

- プログラム、書いてますか？

イントロを書いて

みなさん……

- プログラム、書いてますか？
- プログラム、**書かせてますか？**

イントロを書いて

世はLLMによる大コード生成時代

- Microsoft：リポジトリのコードの 約30%[6, 11]
- Google：新規コードの 30%超 [8]
- 予測：５年で95%[10] / １年で大半 [1]

イントロを書いて

みなさん……

- プログラム、書いてますか？
- プログラム、書かせてますか？
  - ☝️ 世は LLM による大コード生成時代

イントロを書いて

みなさん……

- プログラム、書いてますか？
- プログラム、書かせてますか？
  - ☝️ 世は LLM による大コード生成時代
- では、何を書かせていますか？

イントロを書いて

みなさん……

- プログラム、書いてますか？
- プログラム、書かせてますか？
  - ☝ 世は LLM による大コード生成時代
- では、何を書かせていますか？
  - OCaml？ Rust？ TypeScript？

イントロを書いて

みなさん……

- プログラム、書いてますか？
- プログラム、書かせてますか？
  - ☝️ 世は LLM による大コード生成時代
- では、何を書かせていますか？
  - OCaml？ Rust？ TypeScript？
  - それらの言語は……

イントロを書いて

みなさん……

- プログラム、書いてますか？
- プログラム、書かせてますか？
  ☝ 世は LLM による大コード生成時代
- では、何を書かせていますか？
  - OCaml？ Rust？ TypeScript？
  - それらの言語は……

LLMフレンドリーですか？!?

発表内容を４行で

発表内容を４行で

LLMに書かせる前提の
プログラム言語の
設計について
考える

生成されるコードの品質の問題を挙げて

世はまさに大生成時代

👉 **コード！**

生成されるコードの品質の問題を挙げて

世はまさに大生成時代

👉 **コード！**

- その他
  - スライド
    部分的にそう……プレーンテキストのメモ → LaTeX
  - 画像、動画、音声
    シーンにあった画像が作れて便利

生成されるコードの品質の問題を挙げて

コードがいっぱい生成されてラッキー 😀 と思いきや……

生成されるコードの品質の問題を挙げて

コードがいっぱい生成されてラッキー 😀 と思いきや……

⚠️ 様々な困難が発生 ⚠️

- 💀 構文エラー
- 💀 型エラー
- 💀 実行時エラー
- 💀 一見正しいが仕様を満たさない

生成されるコードの品質の問題を挙げて

コードがいっぱい生成されてラッキー 😀 と思いきや……

⚠️ 様々な困難が発生 ⚠️

- 💀 構文エラー
- 💀 型エラー
- 💀 実行時エラー
- 💀 一見正しいが仕様を満たさない

💸 上記を修復するための反復……

生成コードの品質を上げる既存研究を教えて

コード生成 (合成) 自体は伝統的な分野 [4]

生成コードの品質を上げる既存研究を教えて

コード生成 (合成) 自体は伝統的な分野 [4]

合成手法の一般原則:

- Syntactic basis
  空間を文法・型で絞る
- Oracle-guided search
  オラクルで探索を導く
- Optimization
  候補の選択・順位づけ

生成コードの品質を上げる既存研究を教えて

コード生成 (合成) 自体は伝統的な分野 [4]

合成手法の一般原則:

- Syntactic basis
  空間を文法・型で絞る
- Oracle-guided search
  オラクルで探索を導く
- Optimization
  候補の選択・順位づけ

LLM は？: 品質を上げたい

- **言語内**のガイド
  構文・型
- **言語外**のガイド
  処理系・ツール・skill……
- (LLM 自身のサンプリング)
  temperature, best-of-n

生成コードの品質を上げる既存研究を教えて

言語内のガイド…… syntactic bias の LLM 版？

生成コードの品質を上げる既存研究を教えて

言語内のガイド…… syntactic bias の LLM 版？

- 構文
- 型システム

生成コードの品質を上げる既存研究を教えて

言語内のガイド…… syntactic bias の LLM 版？

- 構文
  曖昧さを減らす、既存言語の構文に寄せる
- 型システム

生成コードの品質を上げる既存研究を教えて

言語内のガイド…… syntactic bias の LLM 版？

- **構文**
  曖昧さを減らす、既存言語の構文に寄せる
- **型システム**
  - 型により生成される候補を絞る [5]
    - 不正なコード片を 75%ほど枝刈り
    - 構文による枝刈りは 9%

  👉 **型=ドキュメント** という営み やはり正しいか

生成コードの品質を上げる既存研究を教えて

言語内のガイド…… syntactic bias の LLM 版？

- **構文**
  曖昧さを減らす、既存言語の構文に寄せる
- **型システム**
  - 型により生成される候補を絞る [5]
    - 不正なコード片を 75%ほど枝刈り
    - 構文による枝刈りは 9%

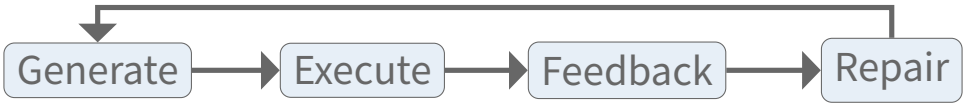
    👉 **型=ドキュメント** という営み やはり正しいか
  - 型システムのリッチさ……
    型エラー情報が増えて嬉しいか？
    Dependent types, linear types, effects ……

生成コードの品質を上げる既存研究を教えて

言語外のガイド……oracle-guided search の LLM 版？

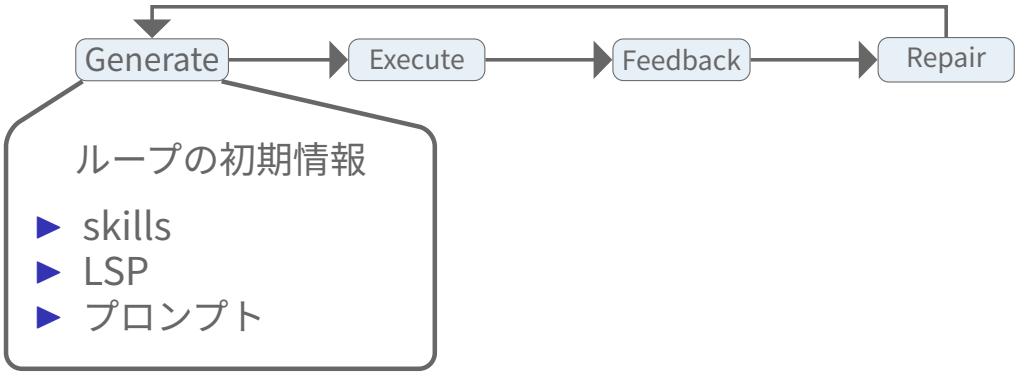
生成コードの品質を上げる既存研究を教えて

言語外のガイド……oracle-guided search の LLM 版？
LLM は self-repair[7] ループで自らコードを生成&修正 [2]



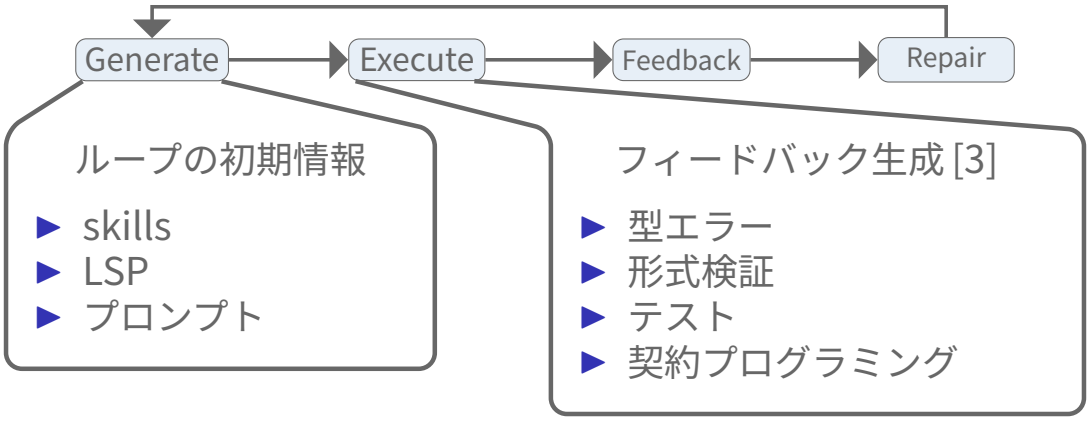
生成コードの品質を上げる既存研究を教えて

言語外のガイド……oracle-guided search の LLM 版？
LLM は self-repair[7] ループで自らコードを生成&修正 [2]



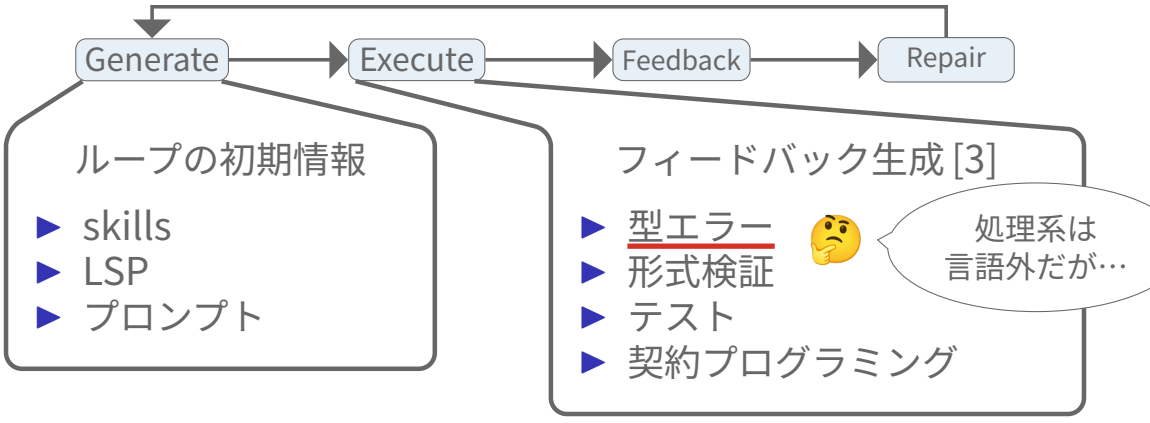
生成コードの品質を上げる既存研究を教えて

言語外のガイド……oracle-guided search の LLM 版？
LLM は self-repair[7] ループで自らコードを生成&修正 [2]



生成コードの品質を上げる既存研究を教えて

言語外のガイド……oracle-guided search の LLM 版？
LLM は self-repair[7] ループで自らコードを生成&修正 [2]



ところで

生成されるコードの品質の問題を挙げて

世はまさに大生成時代

👉 **コード！**

- その他
  - スライド
    部分的にそう……プレーンテキストのメモ → LaTeX
  - 画像、動画、音声
    シーンにあった画像が作れて便利

生成されるコードの品質の問題を挙げて

世はまさに大生成時代

- コード！
  - その他
    - スライド
      部分的にそう……プレーンテキストのメモ → LaTeX
    - 画像、動画、音声
      シーンにあった画像が作れて便利

👉**プログラム言語**っしょ!!!111

LLM 向け言語を作って

ぼくの考えた**最強の**プログラム言語作りは人類の伝統的な営み

LLM 向け言語を作って

ぼくの考えた**最強の**プログラム言語作りは人類の伝統的な営み

これまで

(a) 仕様を考える

(b) 処理系を実装

(c) 周辺ツールを実装

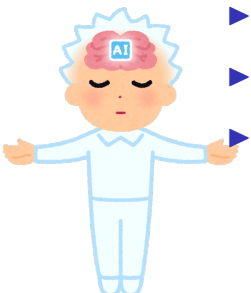
LLM 向け言語を作って

ぼくの考えた**最強**のプログラム言語作りは人類の伝統的な営み

これまで

(a) 仕様を考える
(b) 処理系を実装
(c) 周辺ツールを実装

これから？ w/LLMs



- a だけやって b/c をさせる
- c だけさせる
- a を検証させる

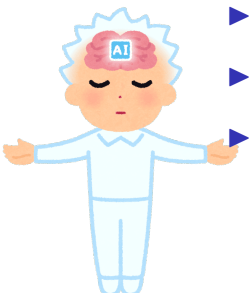
LLM 向け言語を作って

ぼくの考えた**最強**のプログラム言語作りは人類の伝統的な営み

これまで

(a) 仕様を考える
(b) 処理系を実装
(c) 周辺ツールを実装

これから？ w/LLMs



- a だけやって b/c をさせる
- c だけさせる
- a を検証させる

💡 **つまみ食い可能**

LLM 向け言語を作って

ぼくの考えた**最強**のプログラム言語作りは人類の伝統的な営み

これまで

(a) 仕様を考える
(b) 処理系を実装
(c) 周辺ツールを実装

これから？ w/LLMs

- a だけやって b/c をさせる
- c だけさせる
- a を検証させる

💡 **つまみ食い可能**

言語を作るのに
今は(も)**時期が良い！**

LLM 向け言語を作って

世はまさに大 LLM 言語時代……⚔️

LLM 向け言語を作って

世はまさに大 LLM 言語時代……⚔️

- MoonBit … 2023 年から開始したプロジェクト、老舗
- Zerolang … 2026 年 Vercel 製。企業も参入!

LLM 向け言語を作って

世はまさに大 LLM 言語時代……⚔️

- MoonBit … 2023 年から開始したプロジェクト、老舗
- Zerolang … 2026 年 Vercel 製。企業も参入!
- agentlanguages.dev

- 35 言語掲載
  ここにないものも言わずもがな
  動かないものも多少
- リスト中の Vera 言語の作者がホスティング
- ほとんど 2026 年に登場 LLM 言語元年

(とでも言わんばかりだ)

LLM 向け言語を作って

当然……

LLM 向け言語を作って

当然……

俺も作るぞ!!!!

LLM 向け言語を作って

当然……

俺も作るぞ!!!!

ということで
作りました

LLM 向け言語を作って

https://nexus-llm-lang.github.io/latest/

```
import { Console }, * as stdio from "std:stdio"
let main = fn () -> unit require { PermConsole } do
  inject stdio.system_handler do
    Console.println(val: "Hello, Nexus!")
  end
end
```

```
let %h = Fs.open_read(path: "data.txt")
let %r = Fs.read(handle: %h)
let %{ content: text, handle: %h2 } = %r

Fs.close(handle: %h2)
```

```
let @a = compute_a()
let @b = compute_b()
// evaluate parallely
let result = @{ a, b }
```

LLM 向け言語を作って

大まかな設計思想:

- **リテラル上の情報**を増やしたい
  - 全てラベル引数
  - トップレベルは型を明示
  - エフェクトシステム
    - 例外 …継続破棄
    - 要求…継続 1 回
    - (mutli-shot は特別形式)
  - 値の性質に**マーク**をつける
    - linearity %
    - laziness @
    - 参照 &
- ツーリングを厚めに作る
  - 処理系… 型エラー送出
  - MCP
  - LSP
  - skills
- ランタイム安全性
  - WASM ターゲット
    `main` のエフェクト
    →permissions
  - そのための高階エフェクト
    `Fs` → `PermFs`→ `--dir .`

LLM 向け言語を作って

- 人間が仕様を考え LLM が実装
  - 人間…言語仕様、作業指示を出す、バグの修正判断
  - LLM…実装、名前を考える、ロゴを考える
- 現在 self-hosted、ただし bootstrap は Rust 製
- 5 か月ほどで本体が 10 万行、標準ライブラリ 2.8 万行

LLM 向け言語を作って

- 人間が仕様を考え LLM が実装
  - 人間…言語仕様、作業指示を出す、バグの修正判断
  - LLM…実装、名前を考える、ロゴを考える
- 現在 self-hosted、ただし bootstrap は Rust 製
- 5 か月ほどで本体が 10 万行、標準ライブラリ 2.8 万行

😄 冷静になったら出費がやばいので冷静にならない

LLM 向け言語を作って

- 人間が仕様を考え LLM が実装
  - 人間…言語仕様、作業指示を出す、バグの修正判断
  - LLM…実装、名前を考える、ロゴを考える
- 現在 self-hosted、**ただし bootstrap は Rust 製**
- 5 か月ほどで本体が 10 万行、標準ライブラリ 2.8 万行

😄 冷静になったら出費がやばいので冷静にならない

LLM 向け言語を作って

- 人間が仕様を考え LLM が実装
  - 人間…言語仕様、作業指示を出す、バグの修正判断
  - LLM…実装、名前を考える、ロゴを考える
- 現在 self-hosted、**ただし bootstrap は Rust 製** 🤔 ん？
- 5 か月ほどで本体が 10 万行、標準ライブラリ 2.8 万行

😄 冷静になったら出費がやばいので冷静にならない

Fun Facts: 生成される既存言語のコードはすでに高品質

Fun Facts: 生成される既存言語のコードはすでに高品質

LLM 向けの言語で爆コード生成!! と思っていたが

- 構文解析 → 解析･最適化 → コード生成 + 周辺ツール
- LLM に Rust で実装させたらすでにできちゃった！

Fun Facts: 生成される既存言語のコードはすでに高品質

LLM 向けの言語で爆コード生成!! と思っていたが

- 構文解析 → 解析･最適化 → コード生成 + 周辺ツール
- LLM に Rust で実装させたらすでにできちゃった！

😅 最終的に self-hosting は達成できたので、まだ慌てる時間じゃない

Fun Facts: 生成される既存言語のコードはすでに高品質

LLM 向けの言語で爆コード生成!! と思っていたが

- 構文解析 → 解析･最適化 → コード生成 + 周辺ツール
- LLM に Rust で実装させたらすでにできちゃった！

😅 最終的に self-hosting は達成できたので、まだ慌てる時間じゃない

Q. トークン効率は？　　**Q.** 試行回数は言語に依るか？

本気（マジ）になることでしっかり疑問が湧いてくる

ベンチマークをとって

10 言語について**四則演算**、**赤黒木**を LLM に実装させる [*1]

- Ruby
- JavaScript
- TypeScript
- OCaml
- Haskell
- Rust
- **Nexus**
- MoonBit
- Zerolang
- AILANG

青:**Human-Oriented** 言語　赤:LLM 言語

[*1] nexus-llm-lang/benchmark 480 秒内に**生成**できるか、 Claude Code opus-4.8 を使用 Dockerfile で環境分離

ベンチマークをとって

10 言語について**四則演算、赤黒木**を LLM に実装させる [*1]

- Ruby
- JavaScript
- TypeScript
- OCaml
- Haskell
- Rust
- **Nexus**
- MoonBit
- Zerolang
- AILANG

青:**Human-Oriented** 言語　赤:LLM 言語

Nexus と仕様が近い

[*1] nexus-llm-lang/benchmark 480 秒内に**生成**できるか、 Claude Code opus-4.8 を使用 Dockerfile で環境分離

ベンチマークをとって

10 言語について**四則演算**、**赤黒木**を LLM に実装させる [*1]

- Ruby
- JavaScript
- TypeScript
- OCaml
- Haskell
- Rust
- **Nexus**
- MoonBit
- Zerolang
- AILANG

青:**Human-Oriented** 言語　赤:LLM 言語

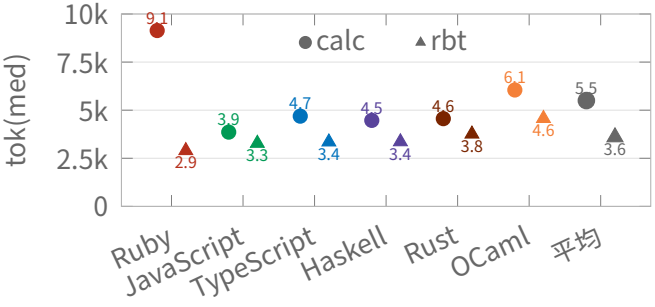
Nexus と仕様が近い

正しいプログラムを生成するまでの過程を検証

- トークン消費数 (**tok**)
- 試行回数
  - 1 発合格か (**pass@1**)
- 5 回やって平均/中央
- 🚫 Web 検索禁止

[*1] nexus-llm-lang/benchmark 480 秒内に生成できるか、 Claude Code opus-4.8 を使用 Dockerfile で環境分離

Human-Oriented 言語

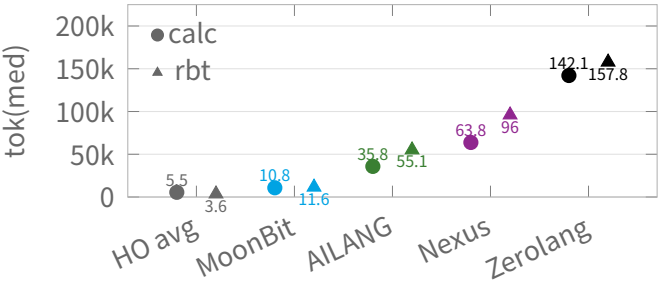


- 全条件 solved 5/5

  学習された言語を測るには簡単すぎ&少なすぎか

		Ruby	JS	TS	Haskell	Rust	OCaml
calc	solved	5/5	5/5	5/5	5/5	5/5	5/5
	pass@1	4/5	5/5	3/5	5/5	5/5	5/5
rbt	solved	5/5	5/5	5/5	5/5	5/5	5/5
	pass@1	5/5	5/5	5/5	3/5	5/5	5/5

HO 言語平均 vs LLM 言語



- LLM 言語も 3/4 言語が solved 5/5
- おなじみのアルゴリズムは未知の構文にも転移する

		MoonBit	AILANG	Nexus	Zerolang
calc	solved	5/5	5/5	5/5	5/5
	pass@1	4/5	5/5	5/5	0/5
rbt	solved	5/5	5/5	5/5	2/5
	pass@1	4/5	3/5	3/5	0/5

ベンチマークをとって

🤨 解答はできたがトークン消費量が……

[*2] ただし 5 回分の合計。ビルド/実行の呼び出しは除いてある

ベンチマークをとって

🤨 解答はできたがトークン消費量が……
1行あたりのトークン:

- 👍 HO 言語 avg 29
- 💀 LLM 言語 380〜660

rbt	HO avg	MoonBit	AILANG	Nexus	Zerolang
LOC	126	181	132	145	270
tok/LOC	29	64	417	662	584

[*2] ただし 5 回分の合計。ビルド/実行の呼び出しは除いてある

ベンチマークをとって

🤔 解答はできたがトークン消費量が……
1行あたりのトークン:
- 👍 HO 言語 avg 29
- 💀 LLM 言語 380〜660

🤔 結局学習量？ ただし**いつ**学ぶか

rbt	HO avg	MoonBit	AILANG	Nexus	Zerolang
LOC	126	181	132	145	270
tok/LOC	29	64	417	662	584

[*2] ただし 5 回分の合計。ビルド/実行の呼び出しは除いてある

ベンチマークをとって

🤔 解答はできたがトークン消費量が……
1行あたりのトークン:

- 👍 HO 言語 avg 29
- 💀 LLM 言語 380〜660

rbt	HO avg	MoonBit	AILANG	Nexus	Zerolang
LOC	126	181	132	145	270
tok/LOC	29	64	417	662	584

🤔 結局学習量？ ただし**いつ**学ぶか
コマンド呼び出しの初手は**いずれも資料の参照**

- 係るトークン消費は 2〜4 割
- `nexus skill`
- `zero skills`
- `ailang prompt`
- HO 言語は**学習済み**
  MoonBit…誕生から 3 年でも可

rbt	HO avg	MoonBit	AILANG	Nexus	Zerolang
コマンド数 [*2]	**0**	**0**	79	678	273

[*2] ただし 5 回分の合計。ビルド/実行の呼び出しは除いてある

ベンチマークをとって

学習効率が良いのは何かな？

- ところで Nexus は stdlib をコードで配布
  学習時に読んでる！
- 類型の AILANG は stdlib をバイナリで配布

学習効率が良いのは何かな？

- ところで Nexus は stdlib をコードで配布
  学習時に読んでる！
- 類型の AILANG は stdlib をバイナリで配布

💡 Nexus で**追加実験**してみよう:

(a)**nexus skill** と (b)**stdlib ソース**の読み込みについて

- (a) あり (b) あり
- (a) あり (b) なし
- (a) なし (b) あり
- (a) なし (b) なし

ベンチマークをとって

(a)	(b)	tok(med)	pass@1
あり	あり	80k	3/5
なし	あり	68k	**4/5**
あり	なし	61k	**1/5**
なし	なし	60k	**1/5**

全条件 solved 5/5

- ソースを読むのは**効果的**
- skill の品質は微妙かも…
- なしなしでも pass5/5 → 型エラーのフィードバックが活きてるか

ベンチマークをとって

calc/rbt/追試より…

- 新言語は skills/examples で事後学習！
- フィードバック情報の量…
  Zerolang は型エラーを 1 検査ごとに 1 つ→ TLE
  伝統的なコンパイラの手法は依然必要

分からんかった…

- 型は? vs untyped langs
  untyped な LLM 向け言語を発見できず…
- フィードバック情報の質

よもやま話で繋いで

リッチ**すぎる**型システム？

- 依存型？
  - LLMs は算術が苦手 [9]
    型検査器にオフロードできるんですかね？
  - Dependent Haskell、答え合わせ出来るか？
- subtyping も少し苦手そう
  適切な粒度……
  - nominal: 関係
  - structural:minimal なフィールドを拡張

既存言語の型エラー再考ブーム、くる？

- LLM **にも**好影響なフィードバックがほしい
- e.g. `GHC.TypeError`
- 詳細な型の埋め込み……（Extensible Effects, Scala Cats, Effect-TS）

early return の優位性

- 深い nest が苦手
- おたく的な気持ち悪さ…
  式指向にしたいけど、return は式ですか？ 文ですか？
- OCaml の `raise` は式 `exn → 'a`
- 継続モナド？

まとめ

- 世はまさに大生成時代…
- **プログラム言語**も生成可能
- 今も言語づくりはアツい!!!
- LLM 向けビリティは open problem
  もちろん人間向けも議論は絶えない
  - 事後学習
  - フィードバック生成
  - 既存の様々な手法
- 😭 この道の研究はお金がかかりまくり

References I

[1] Dario Amodei. *Remarks at the Council on Foreign Relations*. Council on Foreign Relations event. Mar. 2025. url: https://www.facebook.com/councilonforeignrelations/videos/686731397110782/.

[2] Johin Johny Arimbur. *How Many Tries Does It Take? Iterative Self-Repair in LLM Code Generation Across Model Scales and Benchmarks*. arXiv:2604.10508. 2026. url: https://arxiv.org/abs/2604.10508.

[3] Dekun Dai et al. *FeedbackEval: A Benchmark for Evaluating Large Language Models in Feedback-Driven Code Repair Tasks*. arXiv:2504.06939. 2025. url: https://arxiv.org/abs/2504.06939.

[4] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. "Program Synthesis". In: *Foundations and Trends in Programming Languages* 4.1-2 (2017), pp. 1–119. doi: 10.1561/2500000010. url: https://doi.org/10.1561/2500000010.

II

[5] Niels Mündler et al. “Type-Constrained Code Generation with Language Models”. In: *Proceedings of the ACM on Programming Languages* 9.PLDI (2025), pp. 601–626. doi: 10.1145/3729274. url: https://arxiv.org/abs/2504.09246.

[6] Jordan Novet. *Satya Nadella Says as Much as 30% of Microsoft Code Is Written by AI*. CNBC. Apr. 2025. url: https://www.cnbc.com/2025/04/29/satya-nadella-says-as-much-as-30percent-of-microsoft-code-is-written-by-ai.html.

[7] Theo X. Olausson et al. “Is Self-Repair a Silver Bullet for Code Generation?” In: *The Twelfth International Conference on Learning Representations (ICLR)*. 2024. url: https://openreview.net/forum?id=y0GJXRungR.

[8] Gopalakrishnan Palpandi. *Google CEO Sundar Pichai: AI Writes Over 30% of Our Code*. Medium (Bootcamp). Apr. 2025. url: https://medium.com/design-bootcamp/google-ceo-sundar-pichai-ai-writes-over-30-of-our-code-111eb360f272.

III

[9] Ryoma Sato. *LLM のキモい算術*. ジョイジョイジョイ. Oct. 2025. url: https://joisino.hatenablog.com/entry/kimoi.

[10] Kevin Scott. *Microsoft CTO on Where Value Accrues in an AI World: The Future of Software Development*. The Twenty Minute VC (20VC), episode. Mar. 2025. url: https://open.spotify.com/episode/0sYdP2AGQgfLpdGo0UDkUQ.

[11] Kyle Wiggers. *Microsoft CEO Says up to 30% of the Company's Code Was Written by AI*. TechCrunch. Apr. 2025. url: https://techcrunch.com/2025/04/29/microsoft-ceo-says-up-to-30-of-the-companys-code-was-written-by-ai/.