

動的実行情報を用いたLLM逆コンパイルコードの生成手法の提案

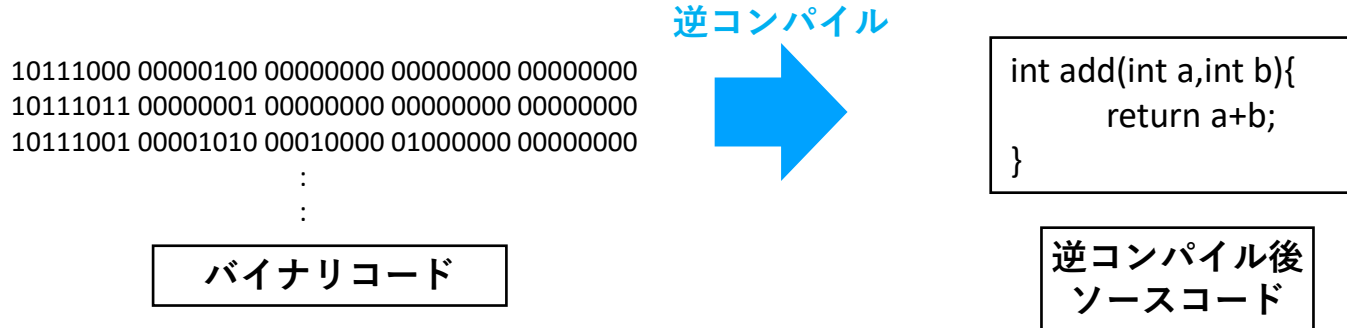
大平研究室

発表者: s2620113 福井遥陽

背景

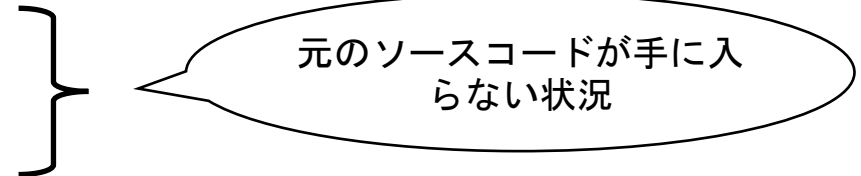
逆コンパイルとは

低水準言語のコードから高水準言語(C,Python,java等)のコードを得る手段.



逆コンパイルの目的

- ・ マルウェアの解析
- ・ ソフトウェアの互換性の維持等

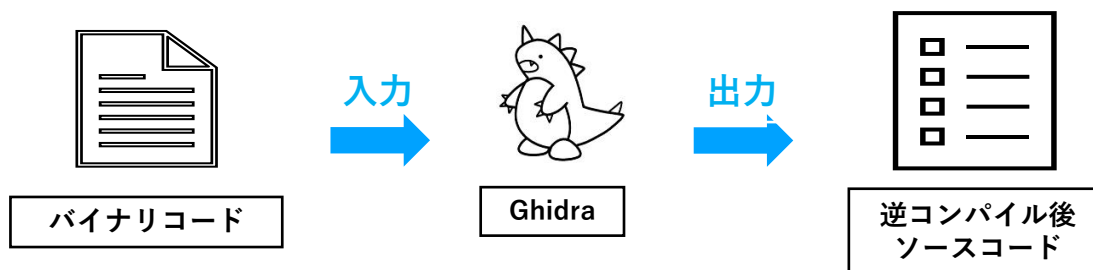


高水準のコードからプログラム全体の処理を追うことが可能となる

既存ツール

現状の逆コンパイルツール

例) Ghidra^[1] (制御フロー解析等に基づくルールベースの逆コンパイルツール)



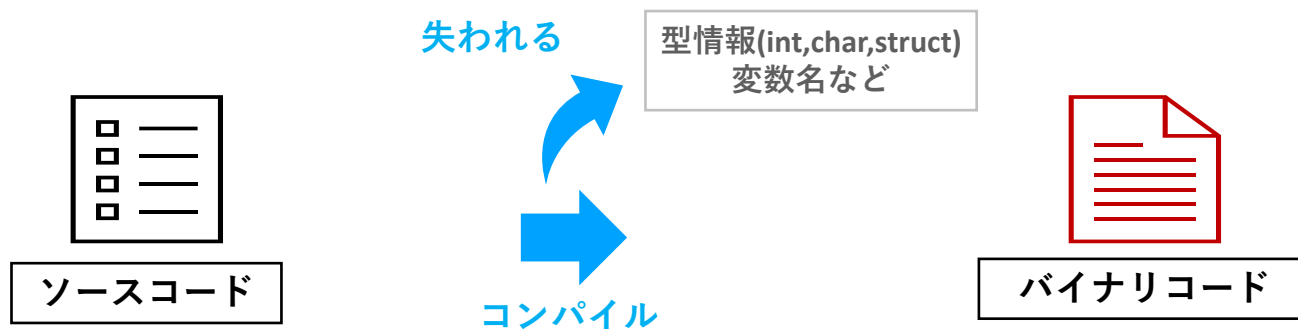
[1] Hanzhuo Tan, Qi Luo, Jing Li, Yuqun Zhang “LLM4Decompile: Decompile Binary Code with Large Language Models”

既存ツール

既存ツールの課題

ルールベースであり、既にコンパイルの段階で失われた意味情報を復元することが困難

例)

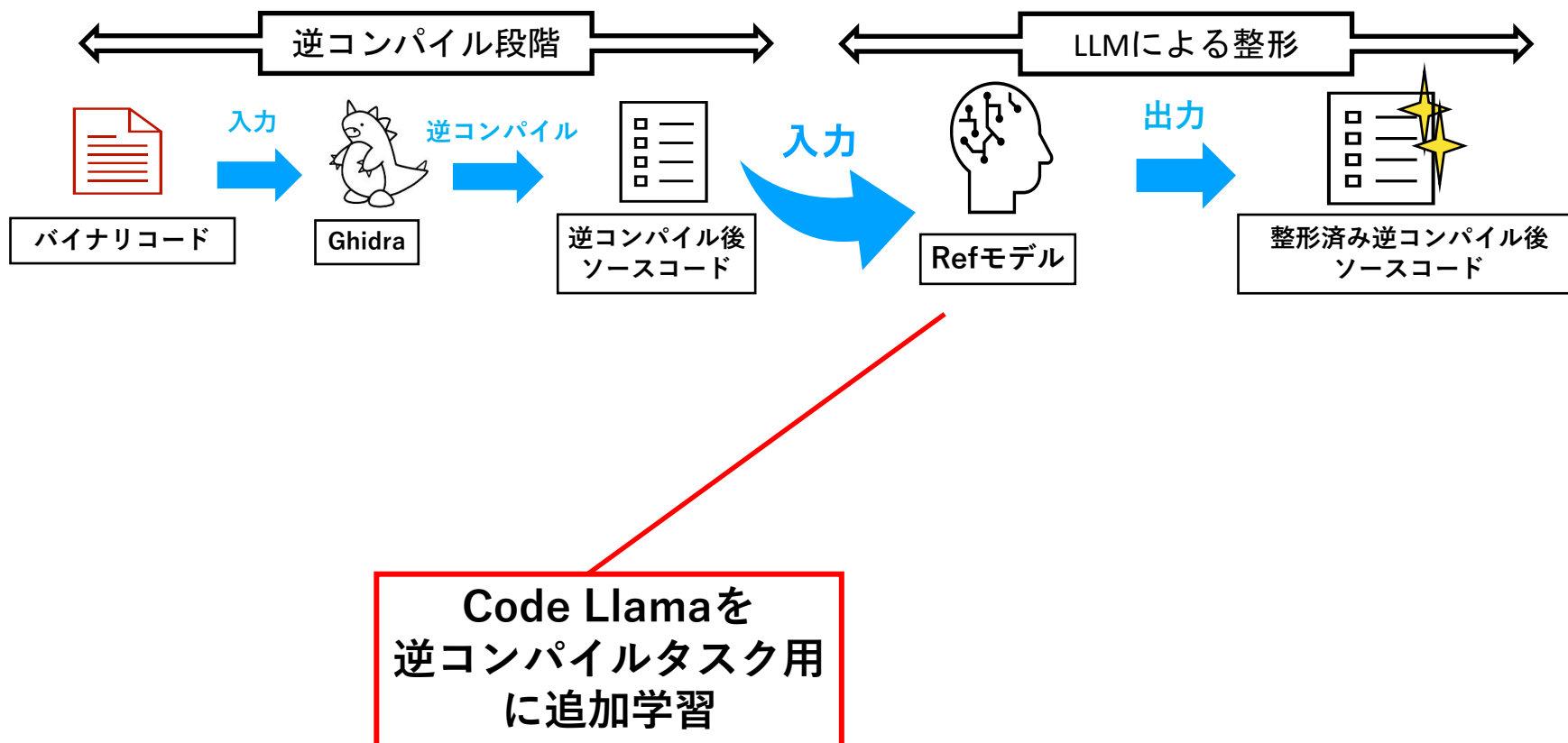


→大規模言語モデル(LLM)で意味情報の補完を試みる手法に遷移

先行研究

逆コンパイル特化LLMモデル

- LLM4Decompile-Ref (Refモデル) [1]



背景

近年の研究

- 制御フロー解析やコンパイル時のエラーメッセージを情報源としたプロンプト設計を試みた研究が行われている.

HELIOS^[5]

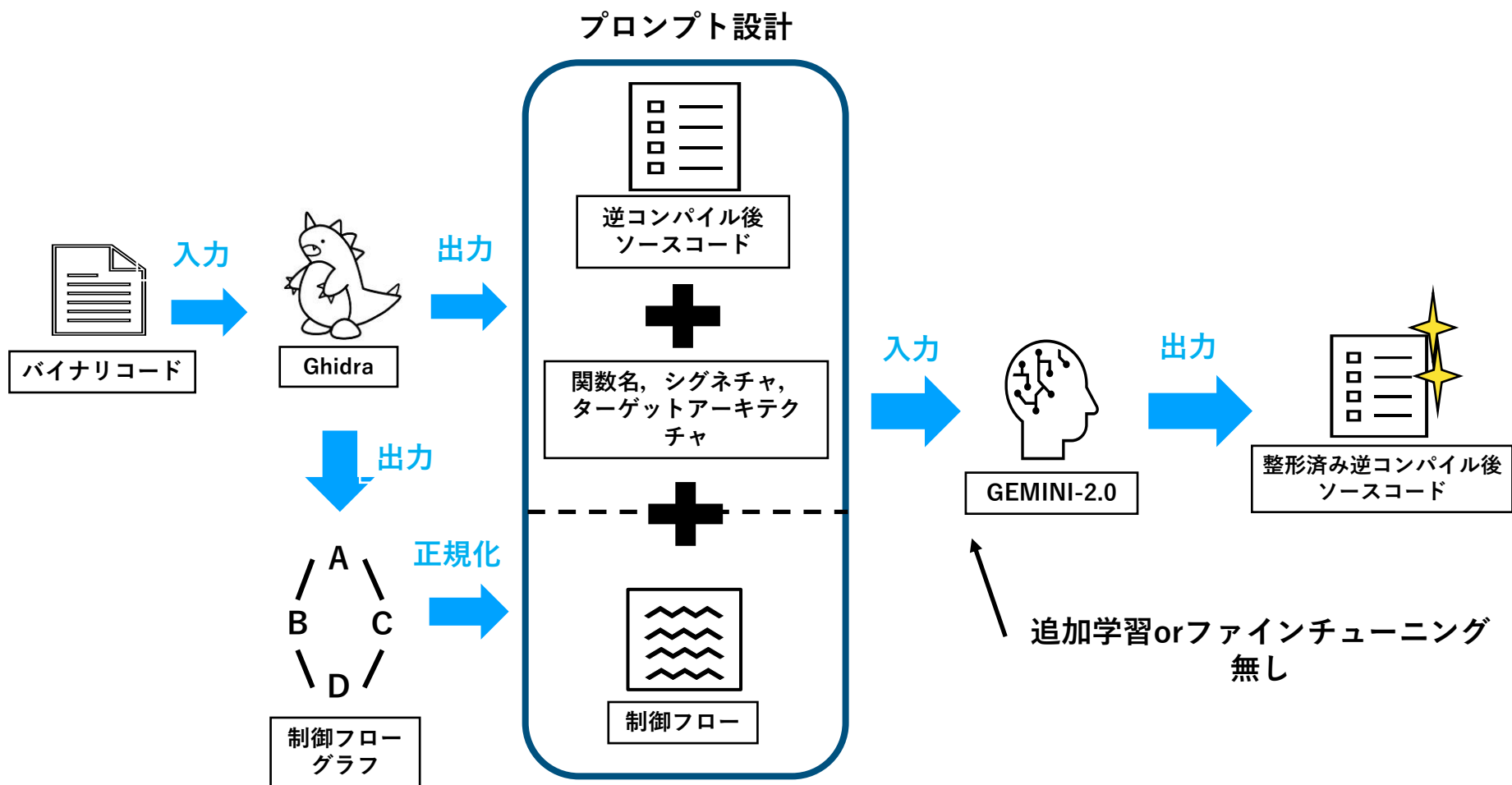
- HELIOSはモデルの追加学習を行わず、静的解析情報をプロンプトとして付加する手法

※現在の逆コンパイル研究における最高峰(SOTA)の研究

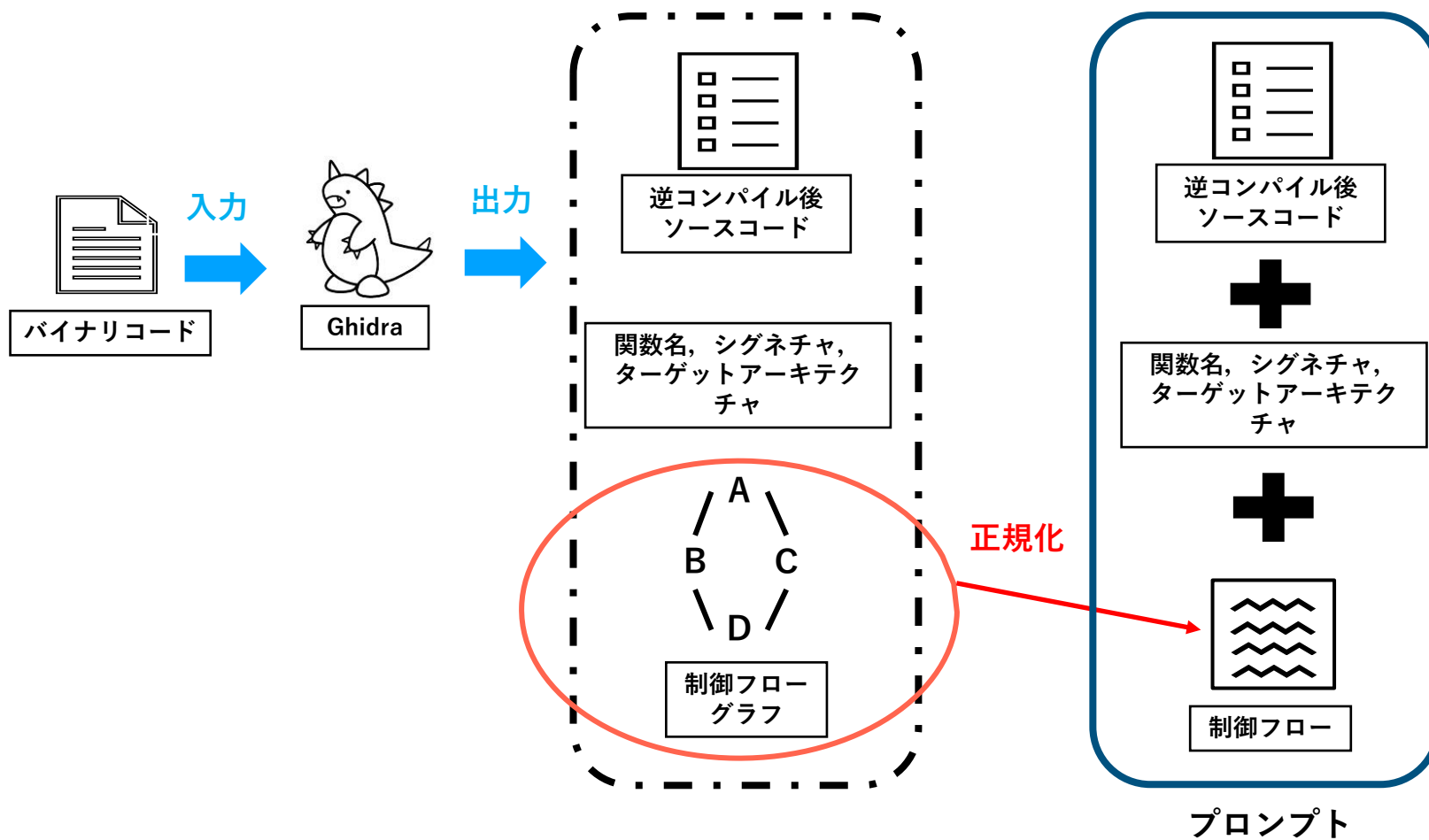
[5] HELIOS: Hierarchical Graph Abstraction for Structure-Aware LLM Decompilation,
Yonatan Gizachew Achamyeleh, Harsh Thomare, Mohammad Abdullah Al Faruque Dept.

HELIOS

フレームワーク



HELIOS



- ここで構成したプロンプトをモデルに入力

HELIOS

貢献

- LLMへの静的解析情報を活用し、再実行可能性の向上に寄与

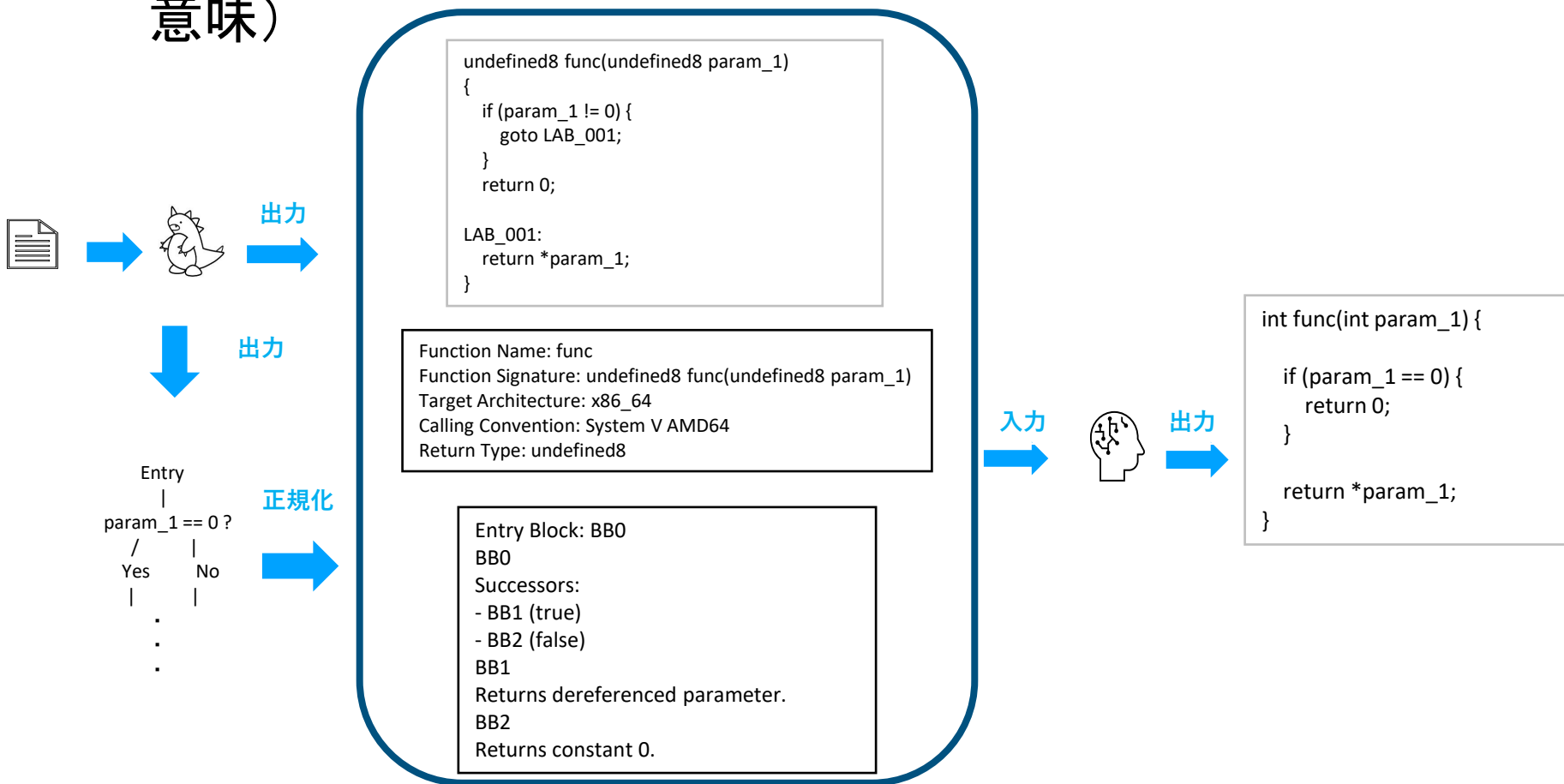
研究の限界

- 再実行可能性は約50%を満たすが、依然として改善の余地あり
- 静的解析のみでは取得できない情報の活用については十分に検討されていない

→ その新しい情報源として実行時トレース情報に着目

問題提起

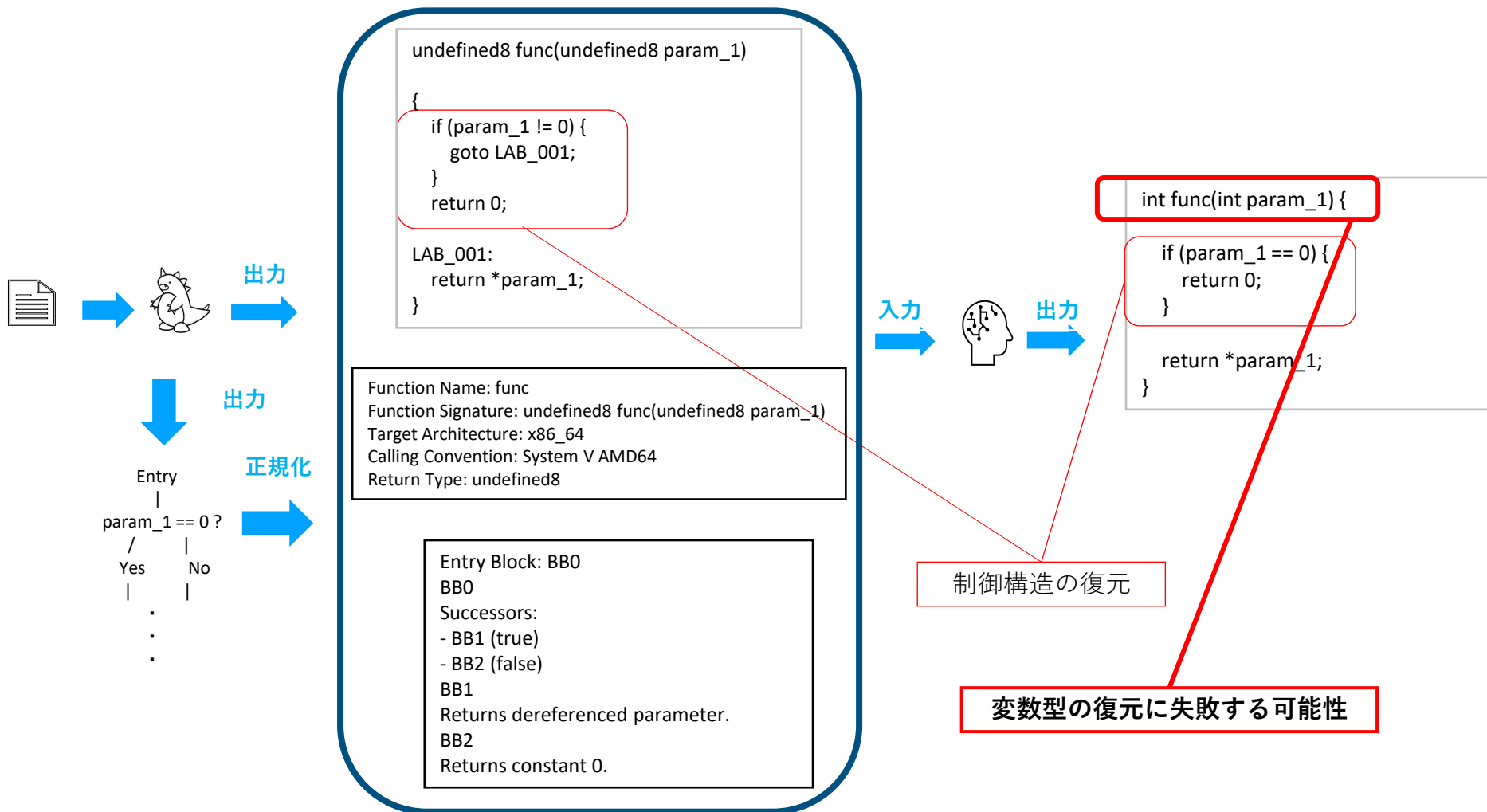
- 静的情報だけでは再実行に必要な情報が十分に復元できない場合がある（変数型・関数シグネチャ・ポインタの意味）



問題提起

※代表例

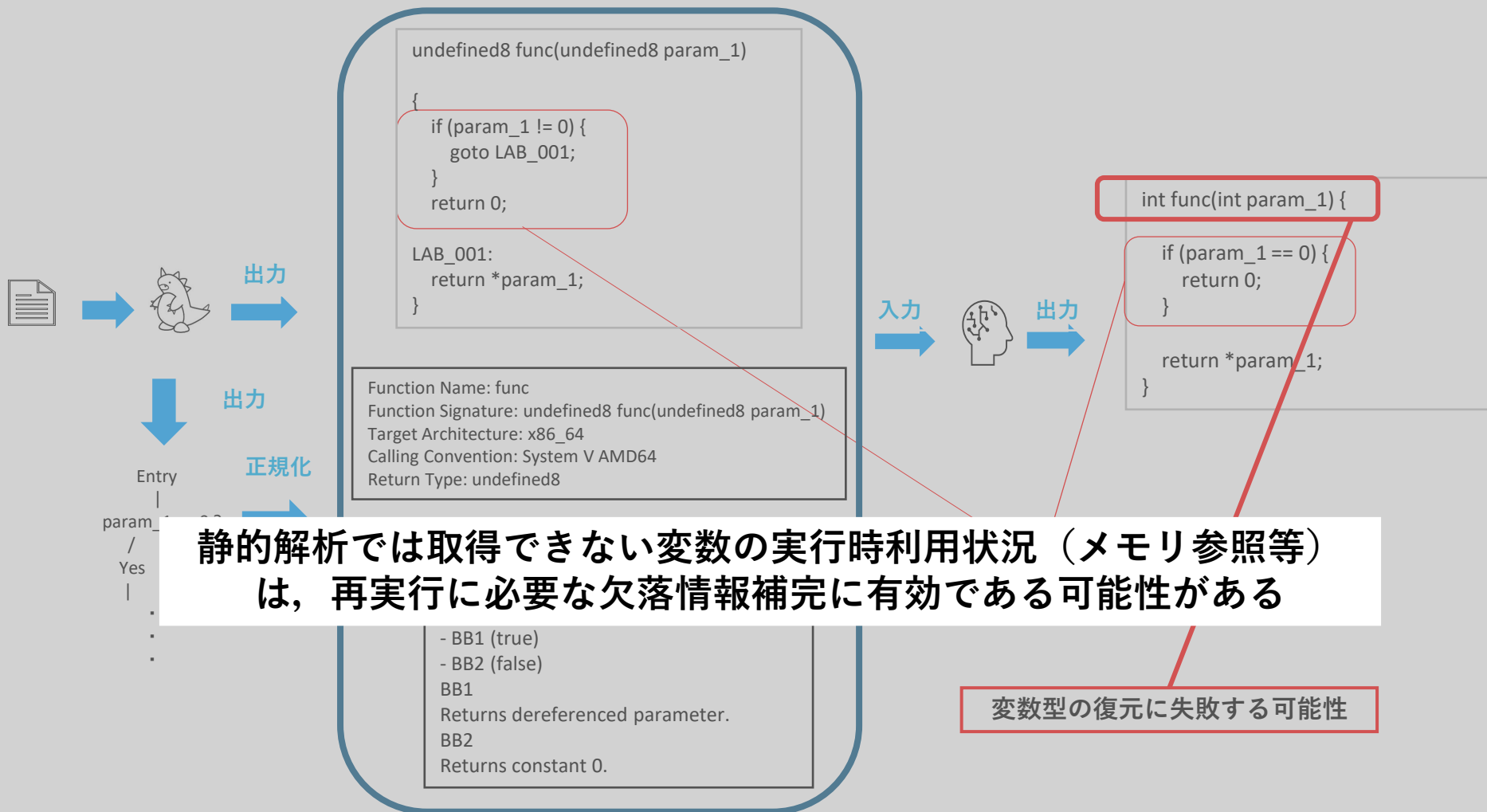
制御構造は復元されたが、変数型が復元できない場合



問題提起

※代表例

制御構造は復元されたが、変数型が復元できない場合



目的

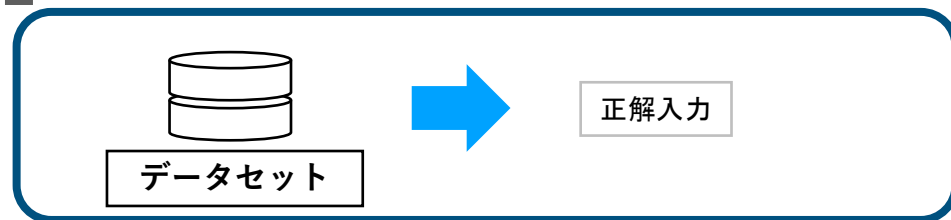
- 取得可能な実行時トレース情報の内、どの情報が再実行可能性の向上に有効なのかを明らかにする

表：取得可能な実行時トレース情報と
期待される効果

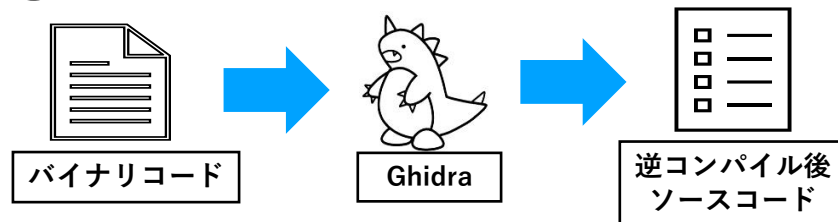
抽出可能な動的情報	再実行可能性への寄与（仮説）
メモリ参照	ポインタ・配列利用の推定
文字列アクセス	文字列型利用の推定
浮動小数点演算	暗黙的な型（float/double）の識別
分岐履歴	正味の実行経路（制御フロー）の特定

提案手法の概要

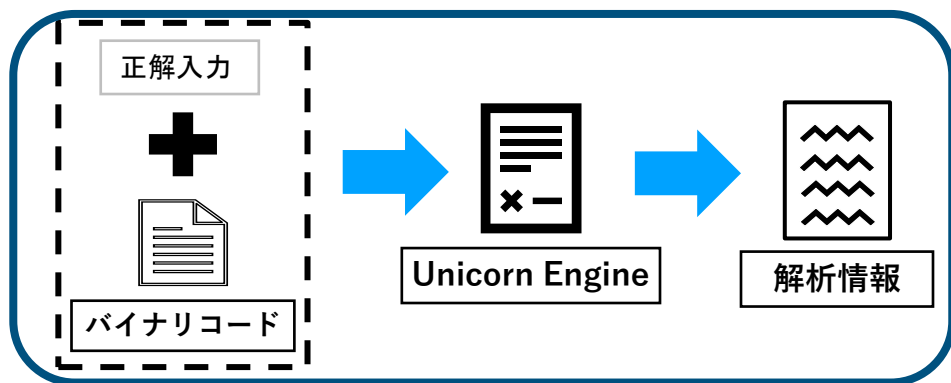
1



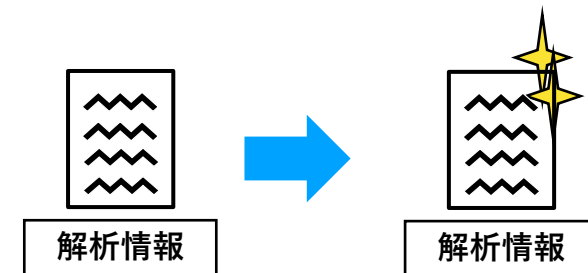
3



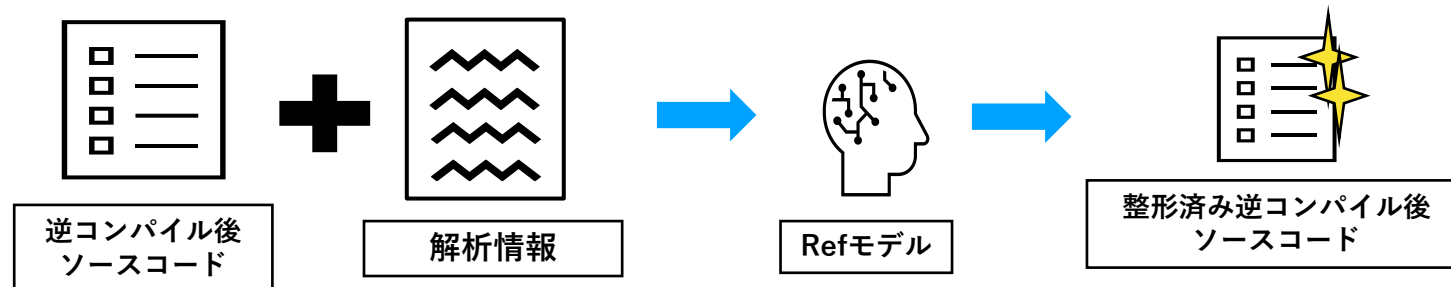
2



4

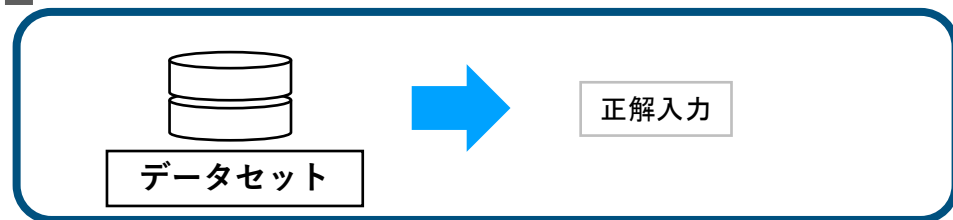


5

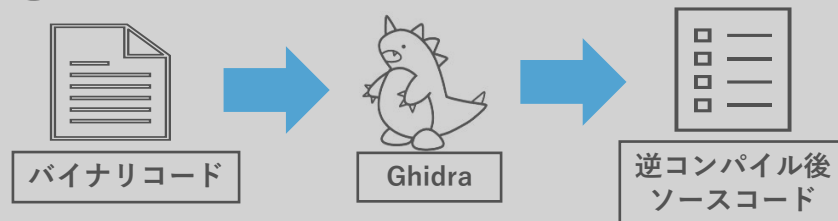


提案手法の概要

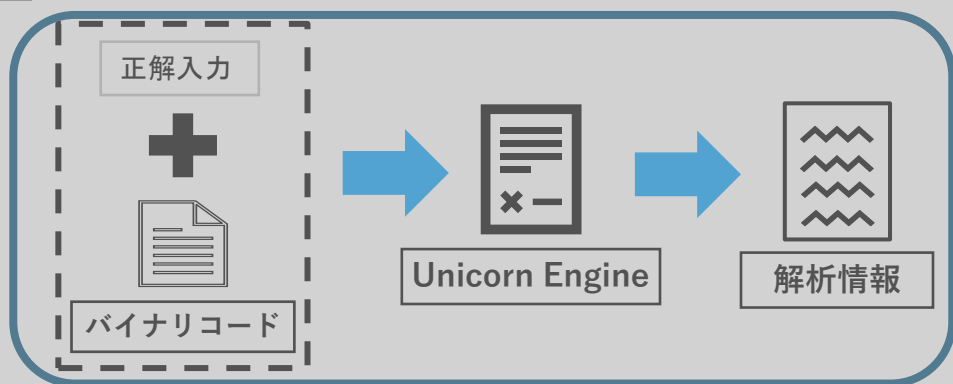
1



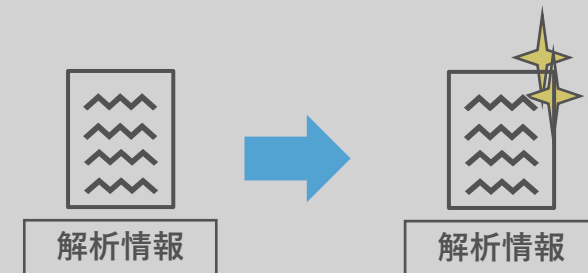
3



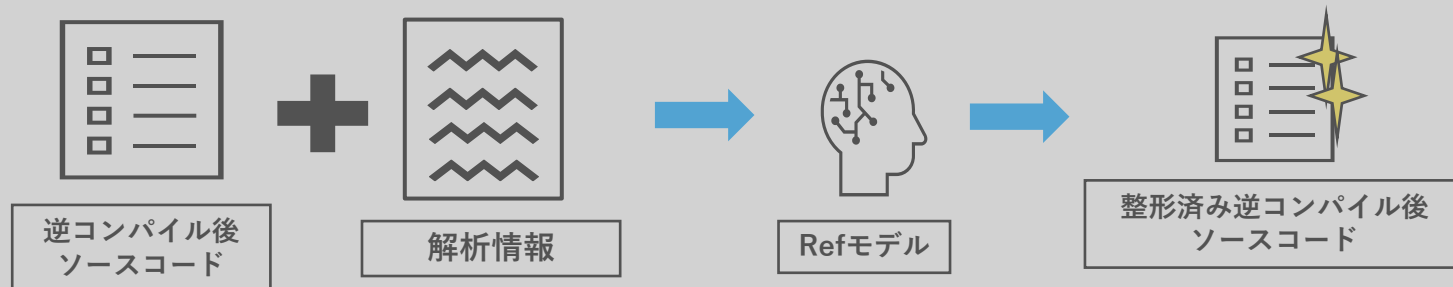
2



4

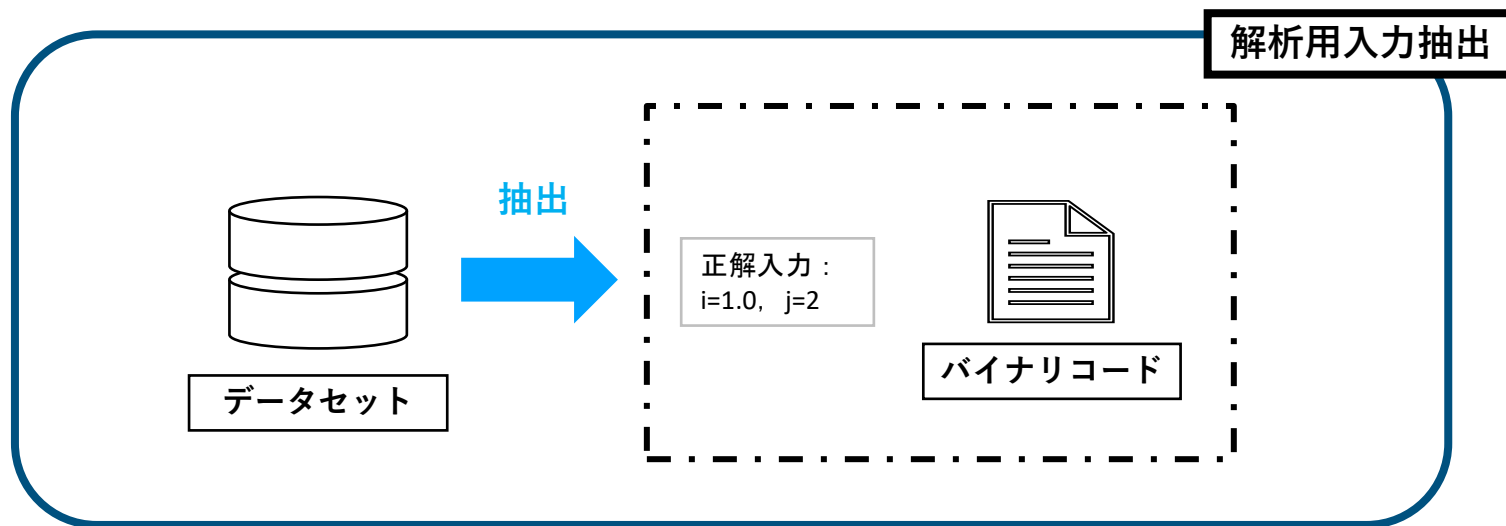


5



提案手法の概要

1.

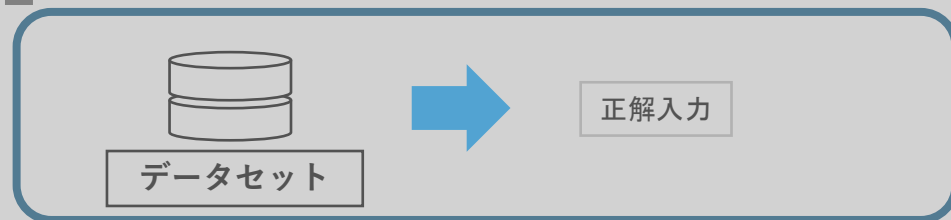


1. 動的解析用のデータ取得

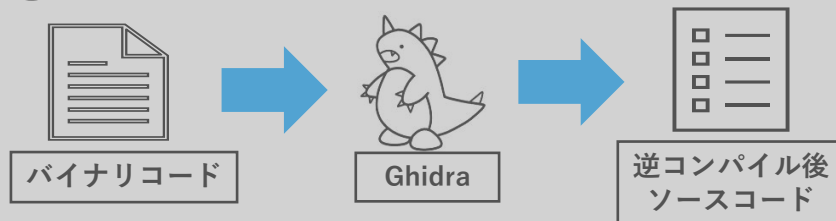
- 入力：データセット (humaneval-decompile)
- 出力：入力用の正解引数＋バイナリコード

提案手法の概要

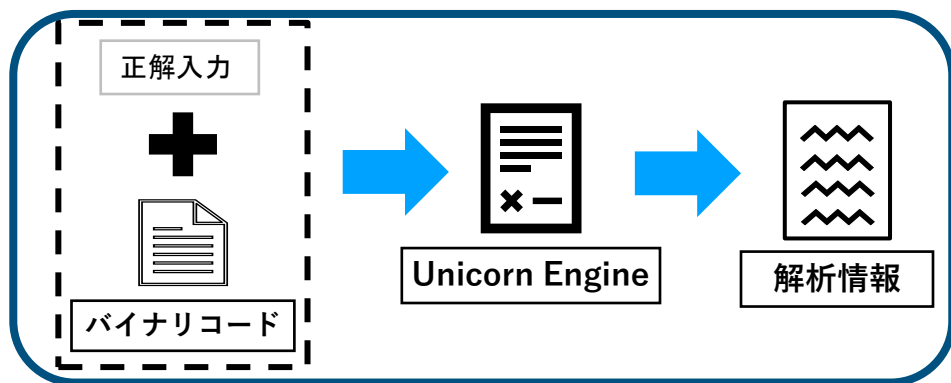
1



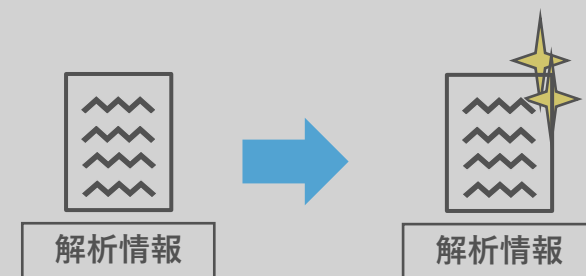
3



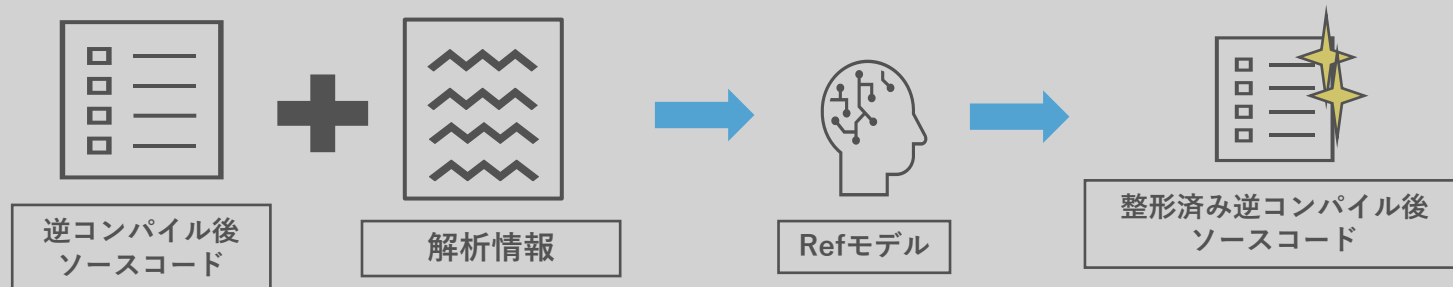
2



4



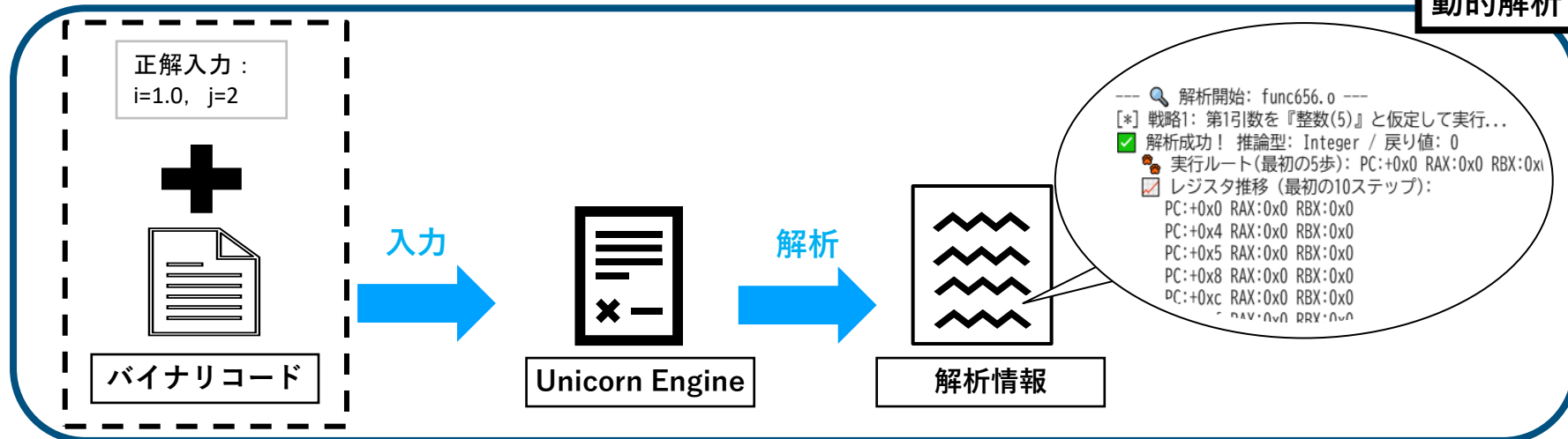
5



提案手法の概要

2.

動的解析

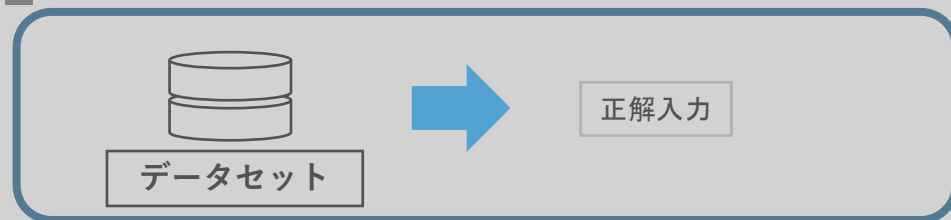


2. 動的解析情報の取得

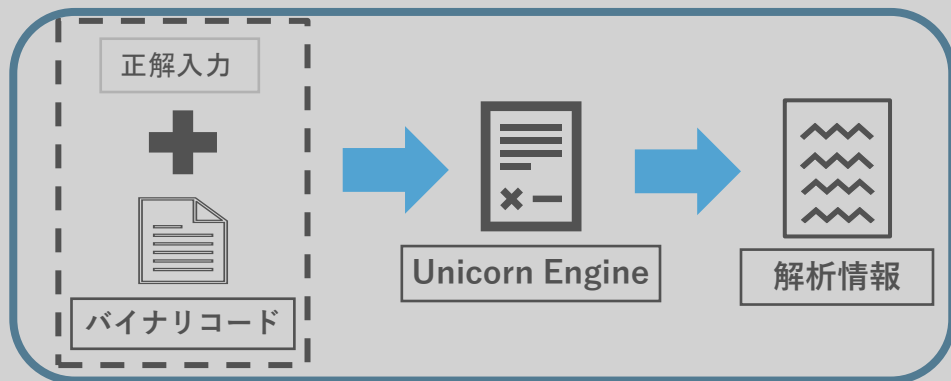
- 入力: バイナリコード+静的解析による正解引数
- 実行: 動的解析ツール(Unicorn Engine)の駆動
- 出力: 解析情報 (メモリアクセス等)

提案手法の概要

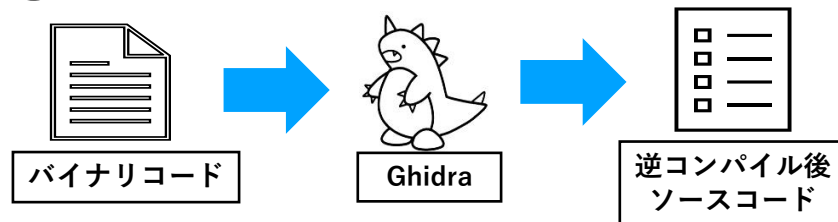
1



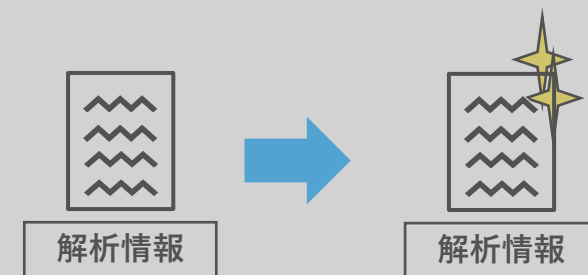
2



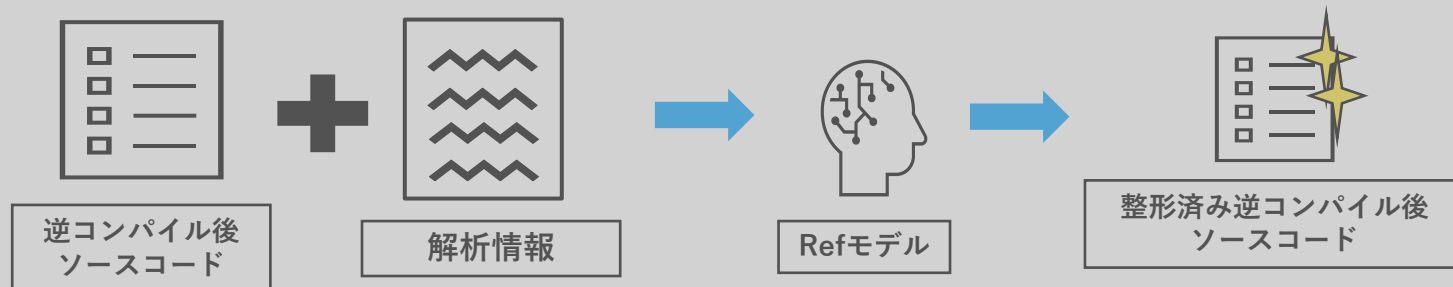
3



4

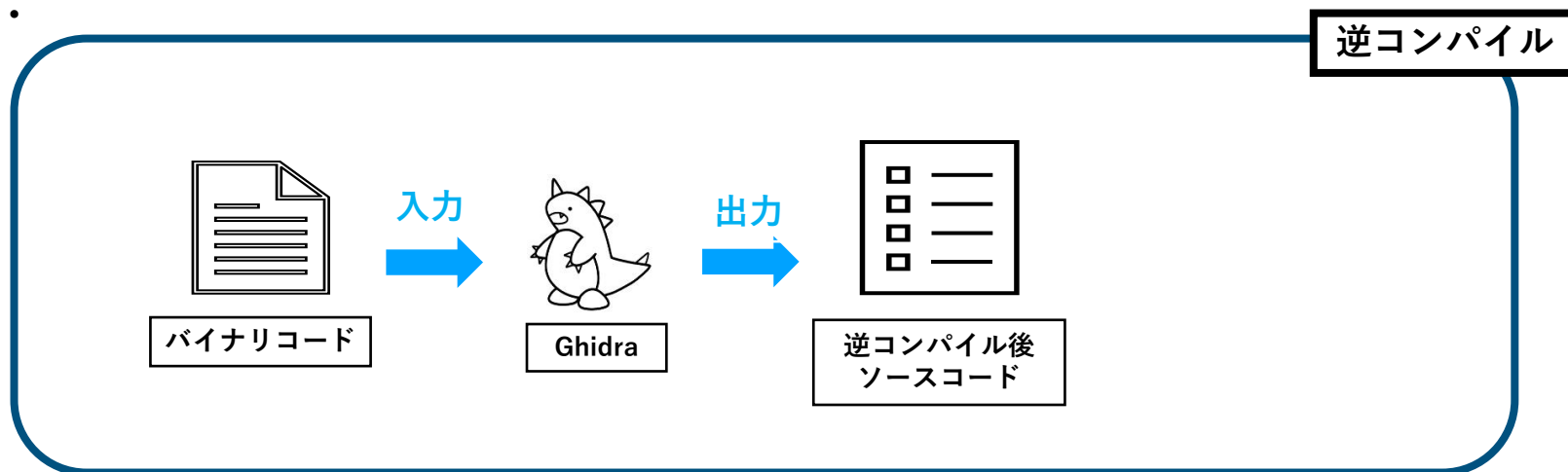


5



提案手法の概要

3.

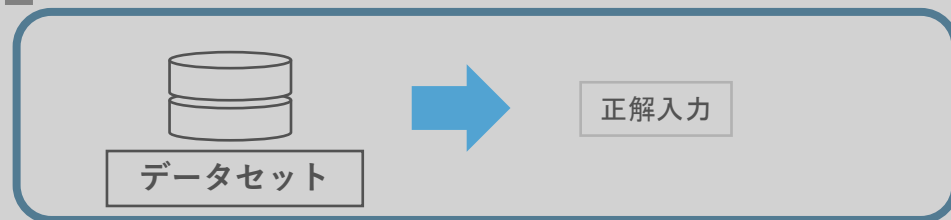


3. 逆コンパイル後ソースコードの取得

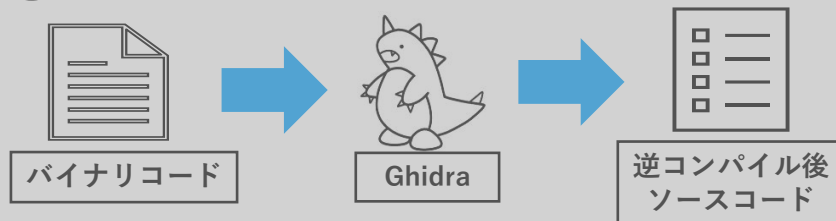
- 入力：バイナリコード
- 実行：Ghidra
- 出力：逆コンパイル後ソースコード

提案手法の概要

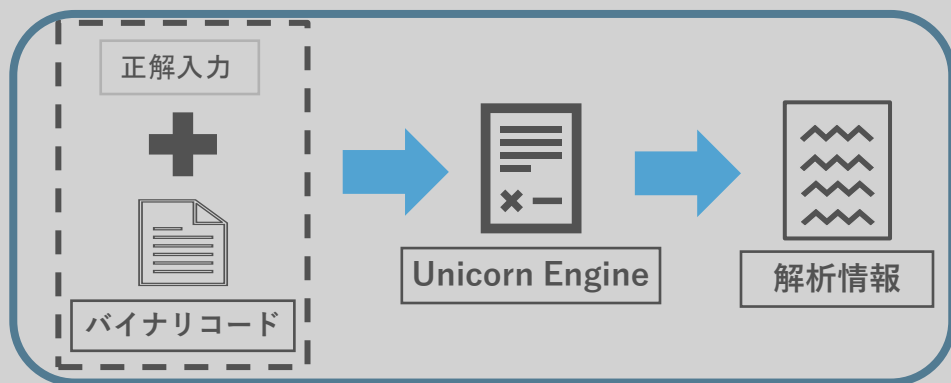
1



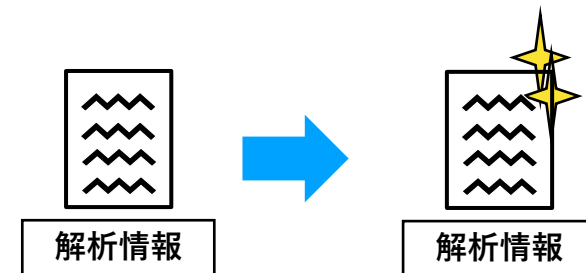
3



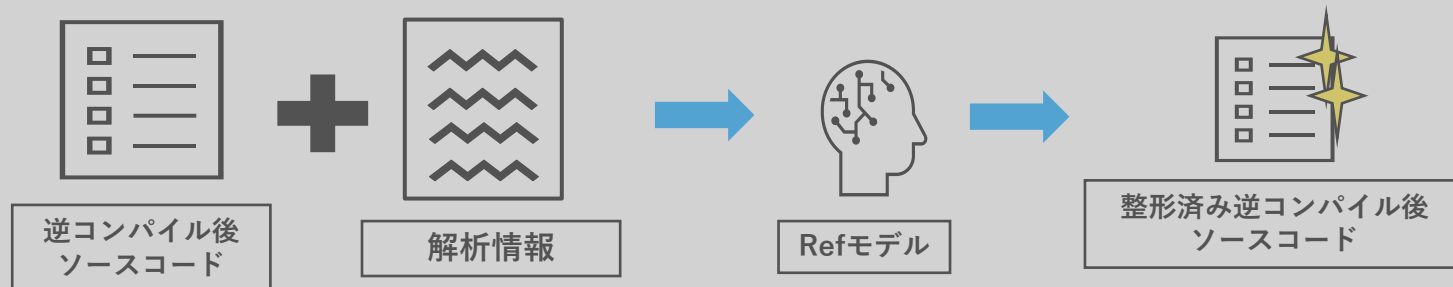
2



4



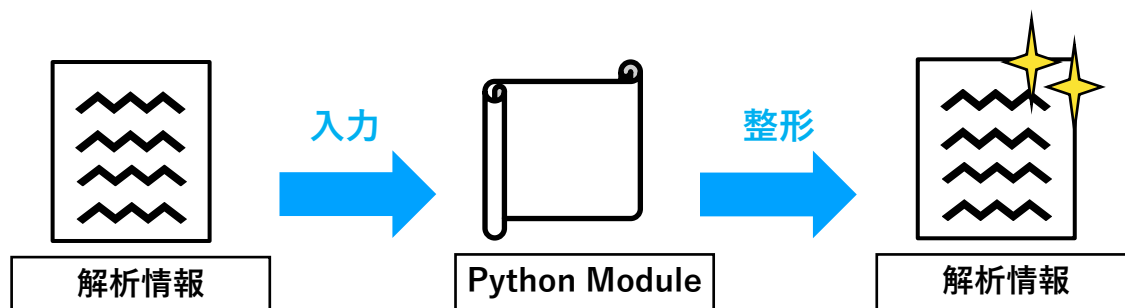
5



提案手法の概要

4.

動的解析

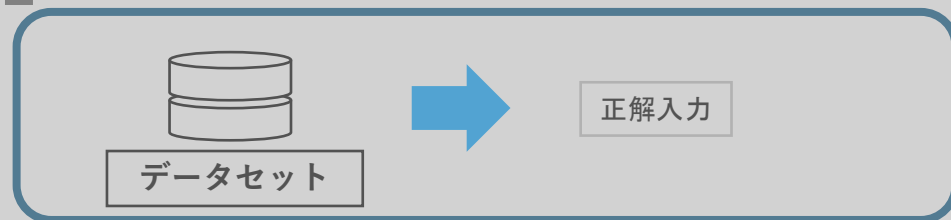


4. 動的解析情報の整形

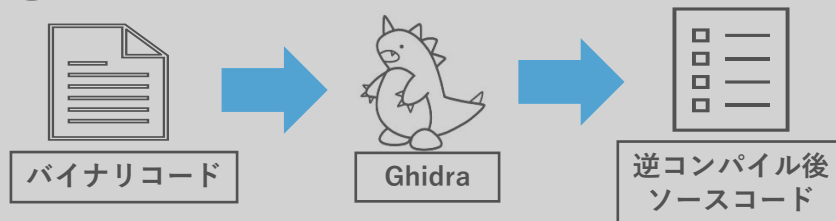
- 入力：解析情報（メモリアクセス等）
- 実行：コンテキスト最適化・RLE前処理モジュール
- 出力：整形された解析情報

提案手法の概要

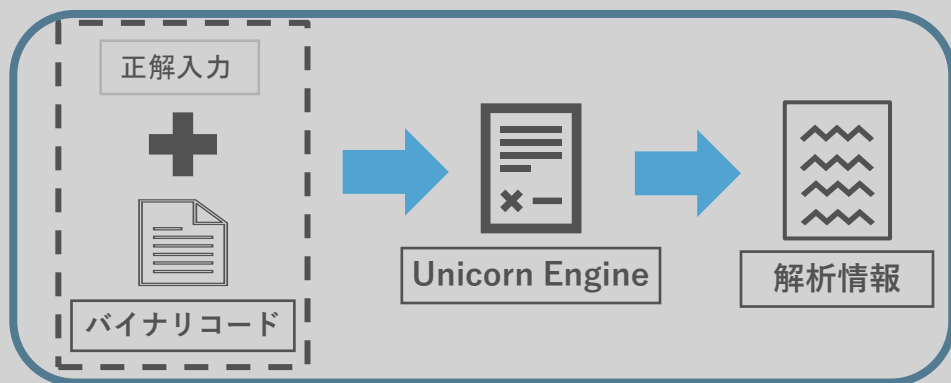
1



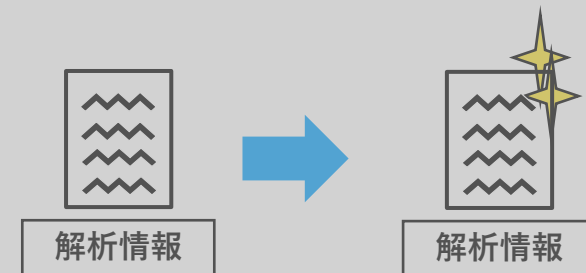
3



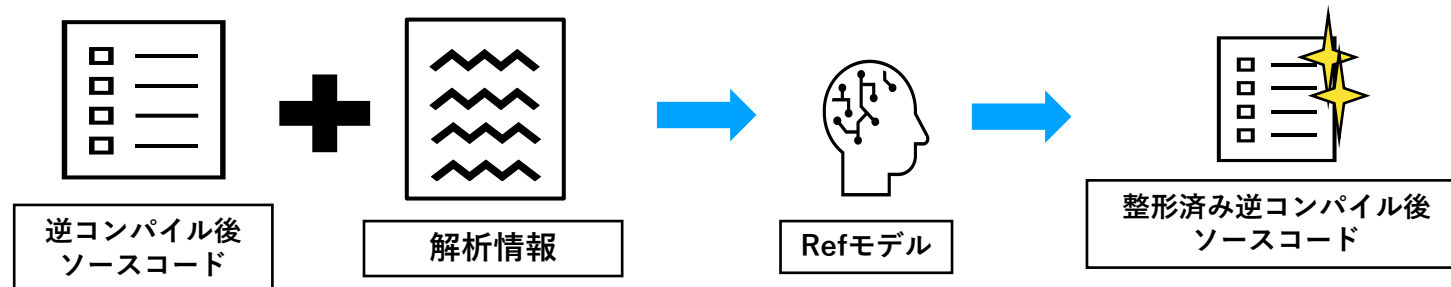
2



4

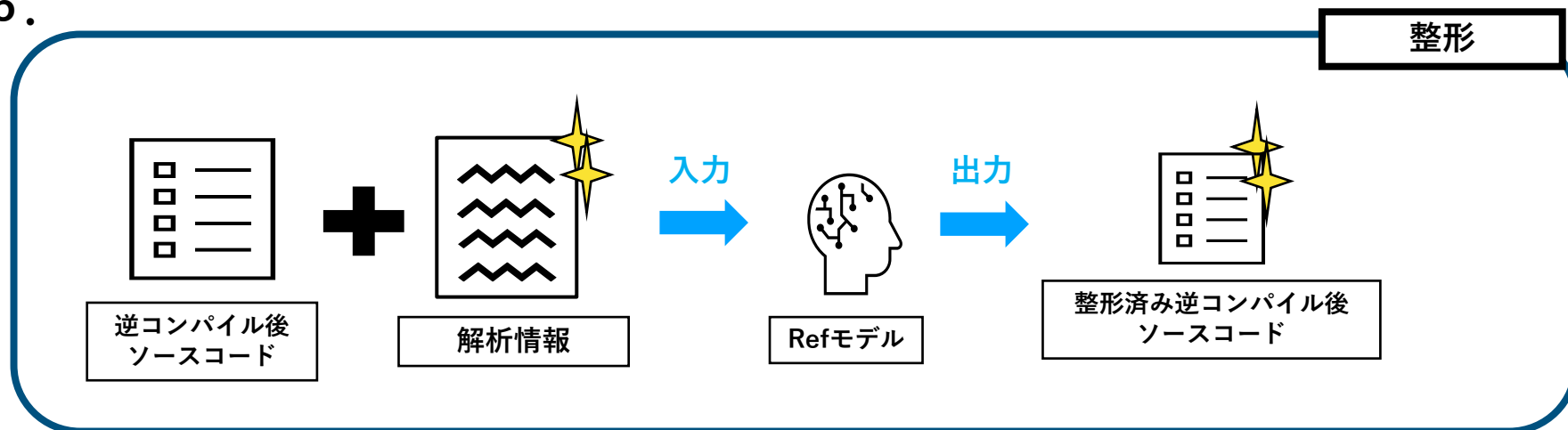


5



提案手法の概要

5.



5. 逆コンパイル後ソースコードの整形

- 入力：整形された各動的解析情報＋逆コンパイル後ソースコード
- 実行：LLM4Decompile-Ref
- 出力：整形済み逆コンパイル後ソースコード

まとめ

本研究の目的

- 動的実行情報の内、どの情報が再実行可能性の向上に有効なのかを明らかにする

今後の方針

- プロンプト設計手法の詳細な提案