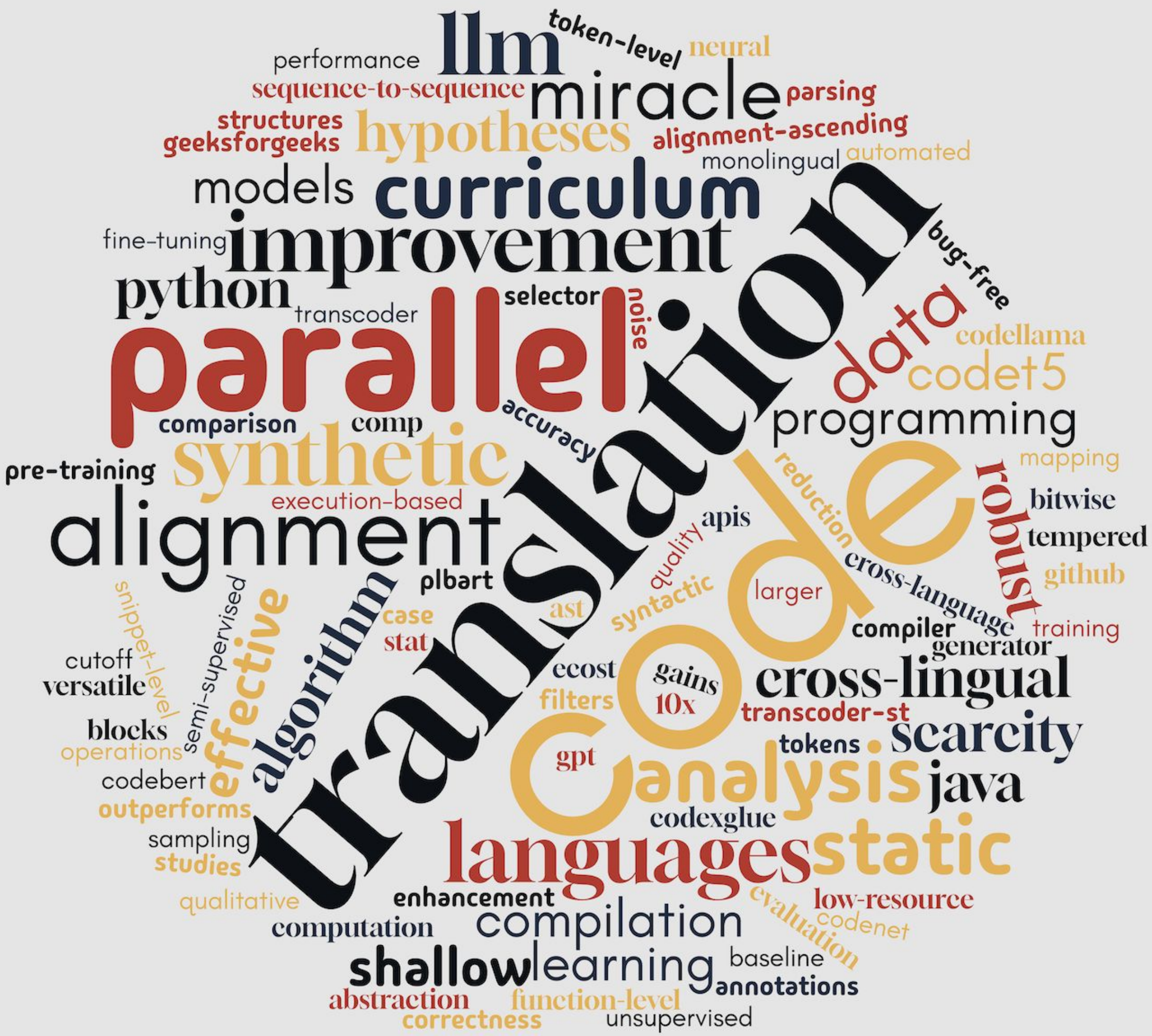


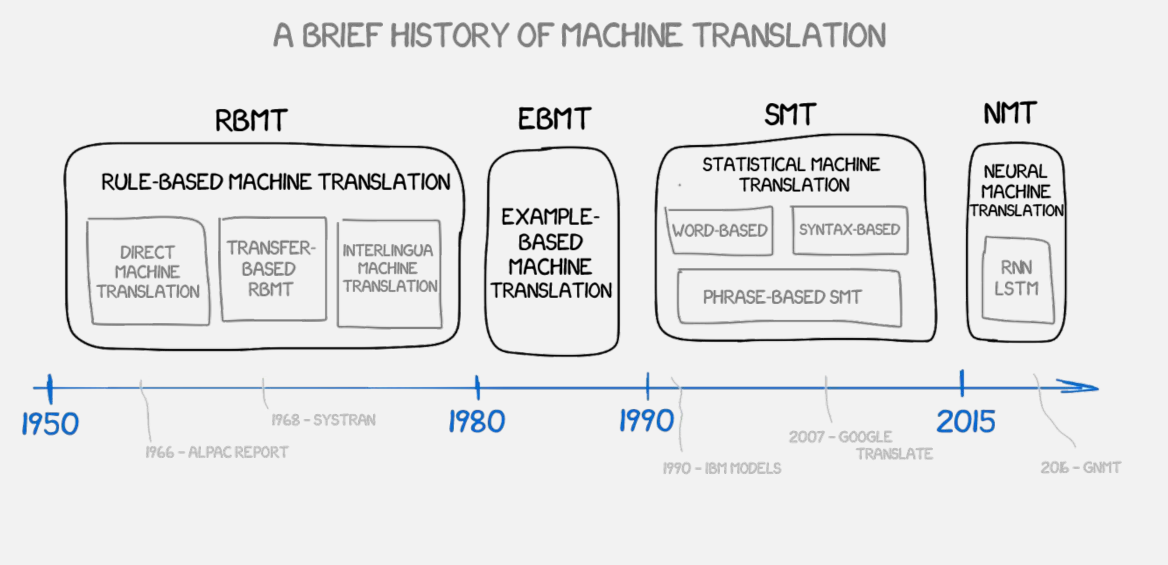
Semi-Supervised Code Translation Overcoming the Scarcity of Parallel Code Data

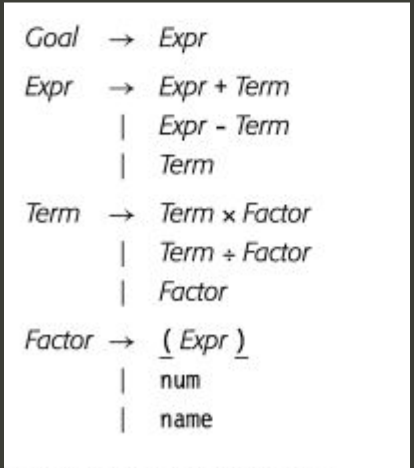
Ming Zhu[1], Mohimenul Karim[1], Ismini Lourentzou[1,2], Danfeng (Daphne) Yao[1]
mingzhu@vt.edu,mohimenul@vt.edu,lourent2@illinois.edu,danfeng@vt.edu
[1]Sanghani Center for AI and Data Analytics, Virginia Tech
Blacksburg, VA, USA
[2] University of Illinois at Urbana-Champaign
Champaign, IL, USA

Presented by: Prayash Joshi, Zehong Wang

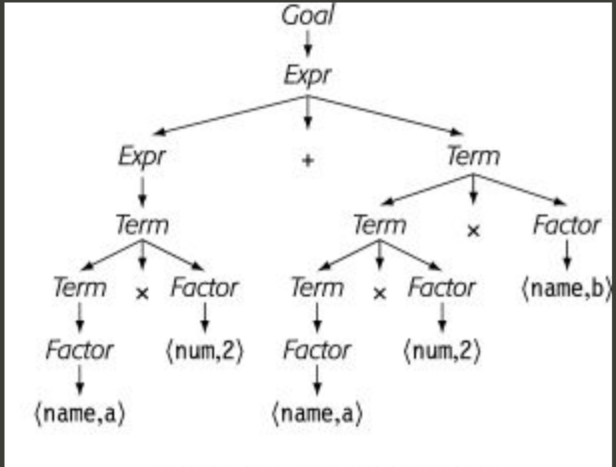
Code Translation

The task of converting source code from one programming language to another.



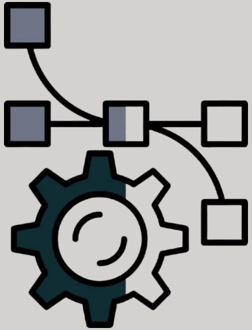


(a) Classic Expression Grammar



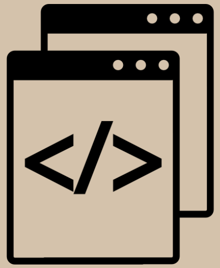
(b) Parse Tree for a × 2 + a × 2 × b

Limitations of Pre-LLM Approaches



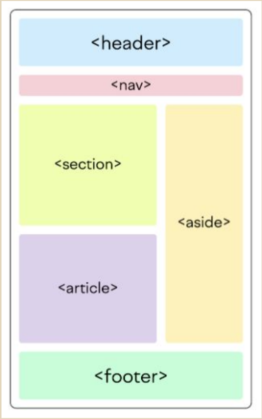
Automation

- Rule-based systems and unsupervised approaches still require non-trivial manual parallel dictionaries



Lack of High Quality Parallel Code Data

- Poor quality dataset leads to less accurate cross-language allignment
- Existing Dataset have limited supported languages & quantity

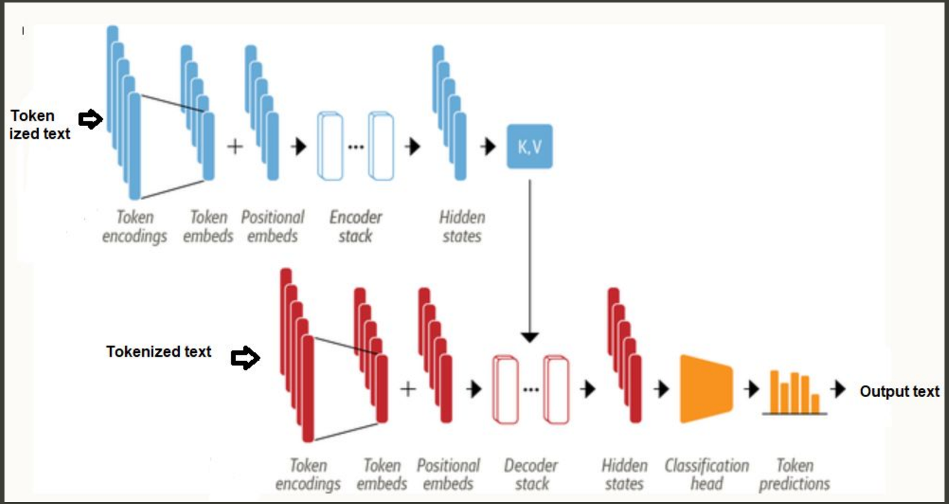


Semantic Gap

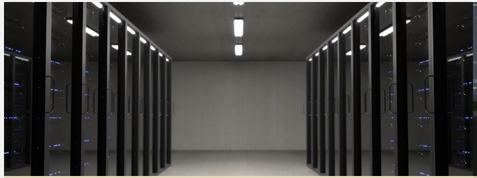
- Existing methods only have token-level consistency, without knowledge of programming structure

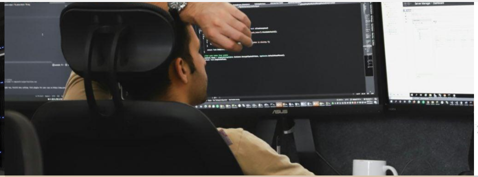
Introduction of LLMs for Code Tranlation

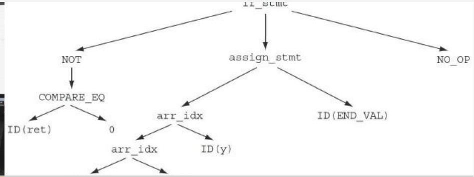
- Encoder-Decoder vs. Decoder Only Models
  - CodeT5+, CodeLLama, etc.
- Extensive pre-training on diverse code corpora with large amounts of source code
  - Strengths in instruction based code generation
- Wide range of supported languages
  - Allows for powerful streamlined automation for code translation



Why is this important?







Code Migration

- Lower cost for migrating existing codebases, legacy code maintenance, and new platform development

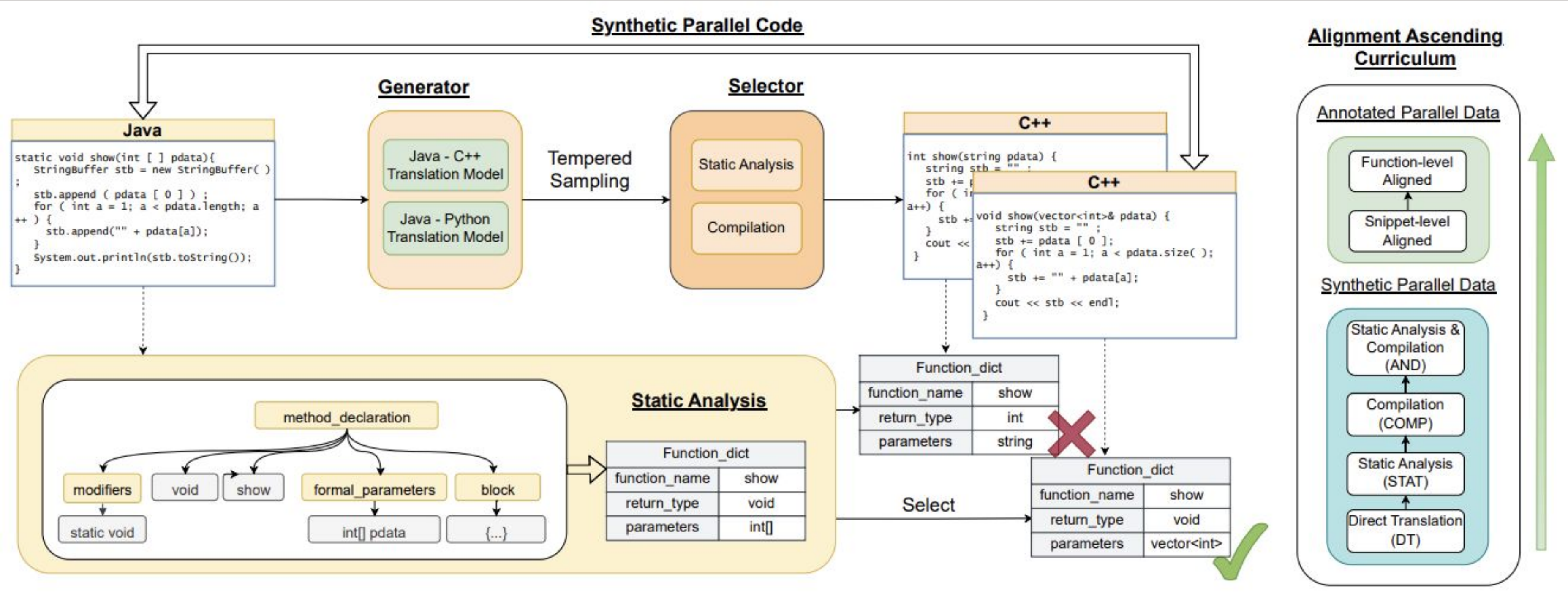
Develop Faster

- Automates manual code porting to allow for development of newer features
- Speed-up in adapting and testing new strategies in different languages

Translation Robustness

- LLMs still lack code correctness guarantees (“Shallow Translations”)
- Novelties in this domain can improvide general code generation with LLMs

Proposed Methodology: MIRACLE



Experimental Setup :

1. Datasets

- **ECoST**: Execution-validated dataset derived from CoST, containing 1,000 function-level pairs for C++, Java, Python, and 150 for C.
- **CodeNet**: Monolingual dataset with 87,000 "Accepted" submissions for synthetic data generation.

2. Baselines

- **Pre-trained Models**: PLBART (139M), CodeT5 (220M), CodeT5+ (770M).
- **Unsupervised Models**: TransCoder (110M), TransCoder-ST (110M).
- **LLMs**: CodeLLama-7B-Instruct, GPT-3.5, GPT-4.

3. Evaluation Metrics

- **Computation Accuracy (CA@1)**: Measures functional equivalence by executing translated code and comparing outputs.

Experiment output example：

Input (Java)	Synthetic Output (C++)	Input (Python)	Synthetic Output (C++)

static void show(int [ ] pdata){
StringBuffer stb = new StringBuffer( ) ;
stb.append (pdata [0]) ;
for (int a = 1; a < pdata.length; a ++) {
stb.append("" + pdata[a]);
}
System.out.println(stb.toString());
}
	void show(vector<int>& pdata) {
	string stb = "" ;
	stb += pdata [0];
	for (int a = 1; a < pdata.size( ); a++) {
	stb += "" + pdata[a];
	}
	cout << stb << endl;
	}
		def f (a , b) :
		p , q = a , b
		while q != 0 :
		p , q = q , p % q
		return a / p * b
			float f (float a , float b) {
			int p = a , q = b ;
			while (q != 0) {
			p = q ;
			q = p % q ;
			}
			return a / p * b ;
			}

public static long gcd (long a , long b){
long c = a % b ;
if (c == 0) {
return b ;
}
else {
return gcd (b , c) ;
}
}
	long gcd (long a , long b) {
	long c = a % b ;
	if (c == 07)
	return b ;
	else
	return gcd (b , c) ;
	}
		def check (n , array) :
		if 1.1 <= n :
		array [0] += 1
		elif 0.6 <= n < 1.1 :
		array [1] += 1
		elif 0.2 <= n < 0.6 :
		array [2] += 1
		else :
		array [3] += 1
			void check (int n , int array [ ]) {
			if (1.0 <= n)
			array [0] ++ ;
			else if (1.0 <= n)
			array [1] ++ ;
			else if (0.2 <= n)
			array [2] ++ ;
			else array [3] ++ ;
			}

Experiment Results:

1. Performance Comparison

- **MIRACLE vs. Baselines**:
    - **C++↔Java**: MIRACLE-CodeT5+ achieves **62.37% CA@1**, outperforming TransCoder-ST (44.09%) and CodeLLama-7B (30.11%).
    - **Python↔C++**: MIRACLE-CodeT5+ reaches **61.75% CA@1**, surpassing CodeLLama-7B (49.70%).
- **Low-Resource Language (C)**:
    - MIRACLE-PLBART improves C→C++ translation by **43%** with only 150 training examples.

2. Curriculum Learning Effectiveness

- Training with DT→STAT→COMP→AND→annotated data boosts CA@1 by **25–30%** across languages.
- Reversing the curriculum order degrades performance, highlighting the necessity of incremental quality.

3. Limitations of LLMs

- **Shallow Translation**: Open-source LLMs mimic source code syntax, causing errors like `deque.empty()` in Python→C++ translations.
- **API Misuse**: LLMs fail to handle language-specific features (e.g., bitwise operations in C).

Discussion Questions

1 | Would you use this tool as Software Engineers? If so, How and When?

2 | Should the generated code follow the license agreement of the original project and ensure compliance during the data generation process?

3 | When the translated code has mistakes and cause significant losses, where should the responsibility be allocated: the model developer, the data provider or the user?

4 | How do you think bias with automated code translation will propagate as we use LLMs for porting large projects to suppport different languages?