In-IDE code generation models

Rebeca Varzaru: D.R.Varzaru@student.tudelft.nl
Responsible Professor: Dr. Fenia Aivaloglou, Supervisor: Xiaoling Zhang

1.INTRODUCTION

Code generation:
- concept → code
- Natural Language Processing (NLP)

Code completion:
- ordered next token suggestions
- most used in-IDE feature
- code faster, avoid typos, explore APIs, reduce keystrokes

2.RESEARCH QUESTION

How have code generation models been integrated into coding environments?

1. What code generation models have been integrated into which coding environments?
2. What techniques have been used for these code generation models?
3. What indicators are used to evaluate code generation models?
4. What aspects should be considered when designing in-IDE code generation models?

3. METHODOLOGY

- Systematic literature review: "research method and process for identifying and critically appraising relevant research, as well as for collecting and analyzing data from said research" [1]
- Search query: ("Large Language Models" OR "code generat*" OR "LLM" OR "code completion") AND ("Coding Environment" OR "ide" OR "Integrated Development Environment" OR "programming environment")
- Platforms: Google Scholar, Scopus, Web of Science

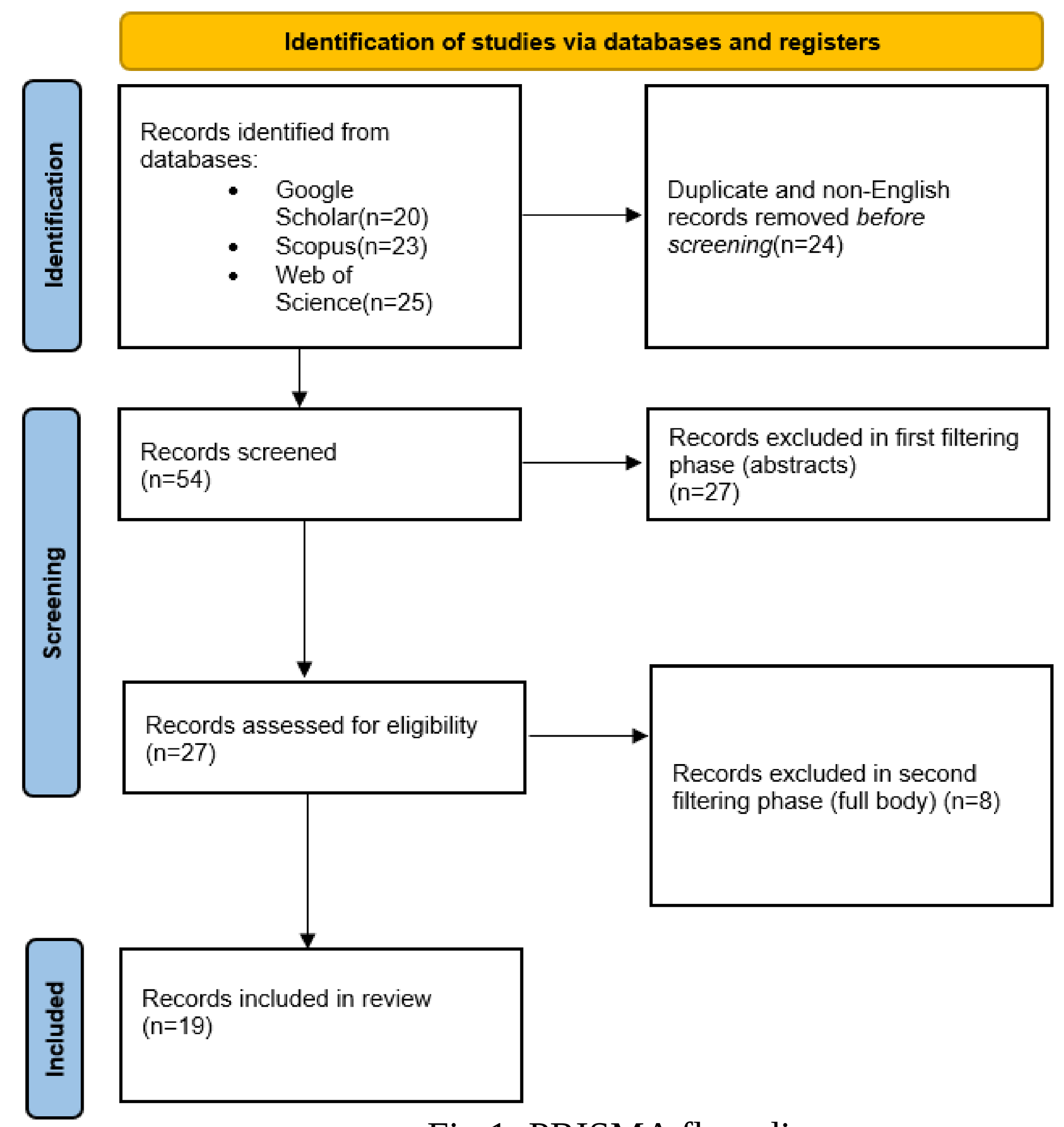

Fig 1: PRISMA flow diagram

4. RESULTS

1.

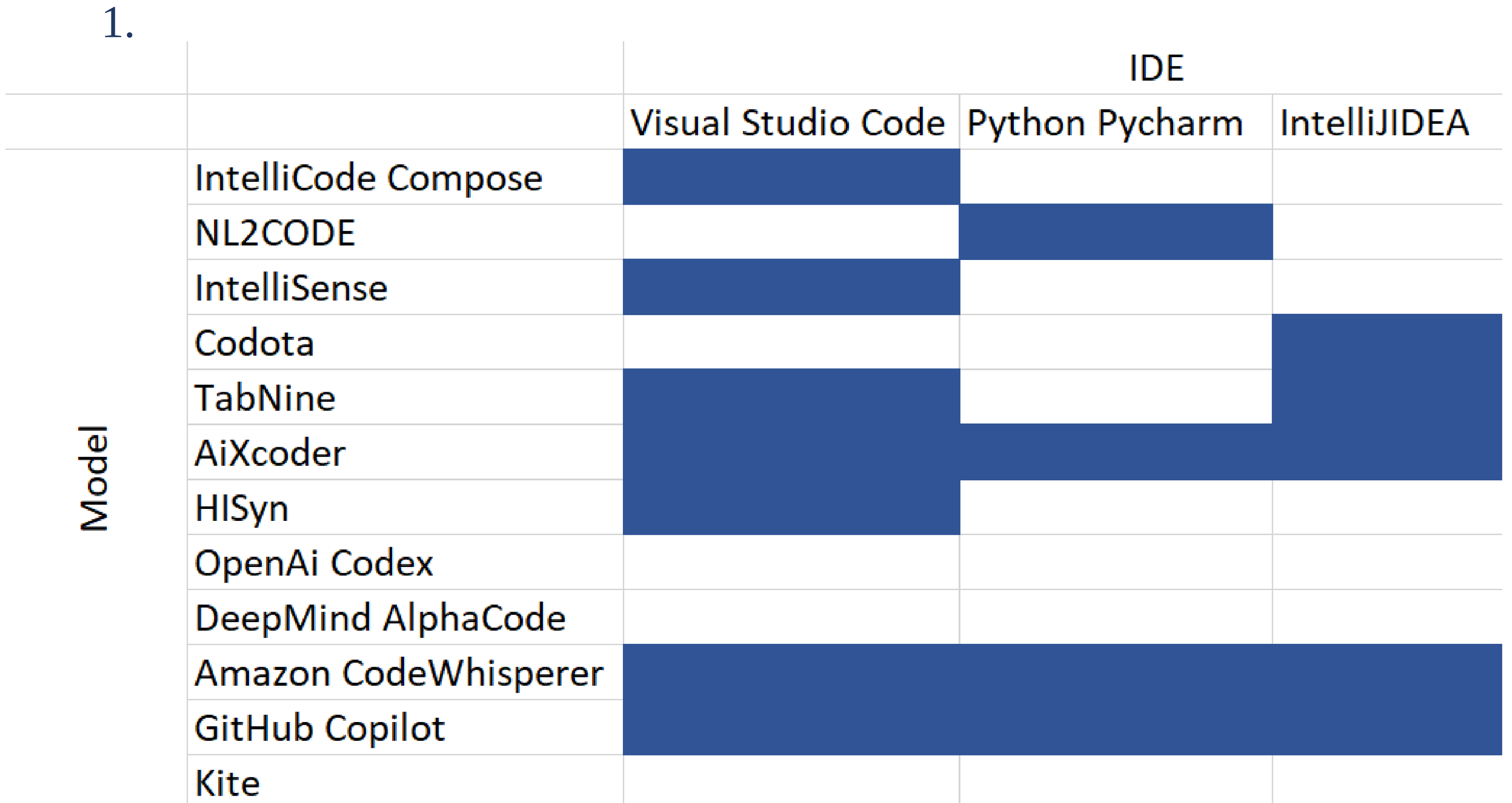

Fig 2: Code generation models and the IDEs where they are integrated

2.
- Generative Pre-trained Transfomer (GPT) – IntelliCode Compose, Codex, Kite
- Tree-based semantic parsing – NL2CODE
- Natural Language Understanding - HISyn

3.

Indicator	Description
Perplexity	How much the model is "surprised" by new data
ROUGE	String similarity between suggestions and target code
Levenshtein similarity	How many edits does it take to transform suggestion into target code
Surfacing Rate (SR)	Total number of completions displayed / number of times a completion could be shown
Click-Through-Rate (CTR)	Accepted completions / total completions
BLEU accuracy score	Token-level overlap between suggestion and reference solution
Accuracy	Fraction of times the correct code is suggested first
Precision	Accuracy of positive predictions
Recall	Completeness of positive predictions
F-measure	Harmonic mean of recall and precision
Top-k accuracy	How often the correct solution appears in the first k recommendations
Mean reciprocal rank (MRR)	Overall rank of the result
Soundness	Syntactical correctness of suggestions
Completeness	Is the suggestion correct and complete enough to provide the desired code snippet
Performance	How fast are the suggestions generated

Fig 3: Indicators used for evaluation

4.
- Code generation should be fast
- All suggestions should be sound and complete
- The generated code should be explainable and provide documentation
- The suggested code segments should be generalizable
- Code generation tools should provide automatic help and guidance for the user and be able to recover from errors
- The tools should be available with as little constraints as possible, such as internet access or high-end technology

5.LIMITATIONS

- Papers only from the last 5 years
- Time constraints
- Single researcher

6.CONCLUSIONS

- Growing trend in AI-driven code generation
- Most popular underlying model seems to be GPT
- Emerging challenge – teaching users to use Natural Language (NL) prompts effectively
- Future work – a more comprehensive literature review without the time constraints; implementation of code generation model following guidelines from RQ4

7. REFERENCES

[1] Khalid S Khan, Regina Kunz, Jos Kleijnen, and Gerd Antes. Five steps to conducting a systematic review. Journal of the royal society of medicine, 96(3):118–121,2003