A Study on Improving LLM-based Code Completion Using LR Parsing

Atique Md Monir Ahammod Bin

Department of Artificial Intelligence Convergence

02 June 2025

Software Languages and Systems Laboratory for Security

전남대학교
CHONNAM NATIONAL UNIVERSITY

Outline

- Introduction
- Contributions
- Experiment
- Results and Discussion
- Implementation
- Related Work
- Conclusion

Introduction

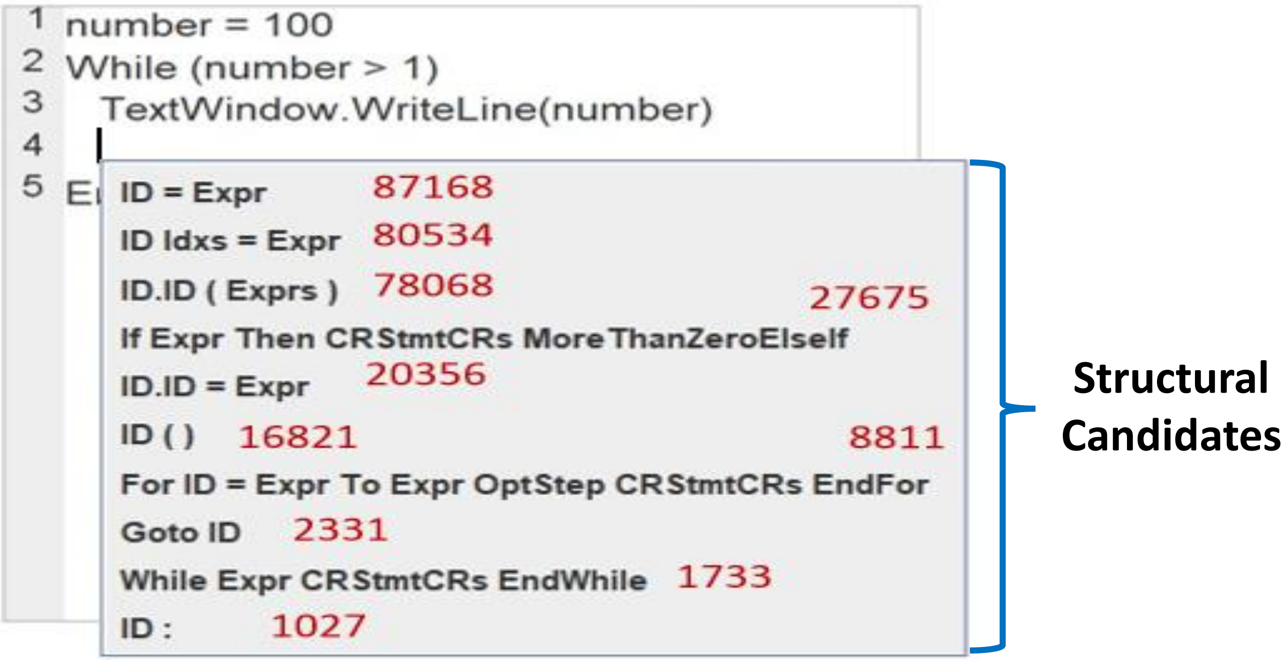
Introduction

Code completion or **autocomplete** is a crucial feature in modern IDEs.

- Suggests code based on current context and language syntax.
- Reduces typing time, detects syntax errors early, and boosts productivity.
- Traditional systems: prefix filtering & static ranking.
- Recent research: LR-parsing-based approaches.

Limitations of previous LR-parsing based approaches:

- Suggest only structural candidates
- Require manual refinement to complete code
- Leads to usability challenges



LR-parsing Based Code Completion in Microsoft Small Basic [ACM SAC 2024]

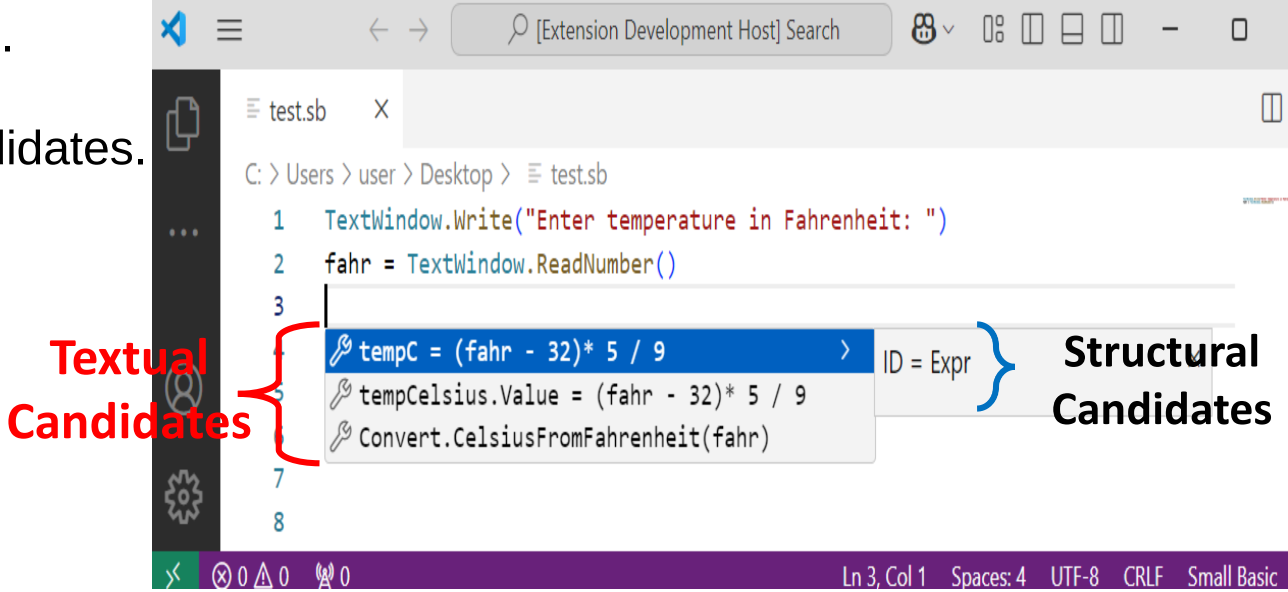
Introduction

Integrate the generative capabilities of Large Language Model (LLM)

- ChatGPT widely used assisting coders in code completion tasks.
- Our approach focuses on fine-grained code completions, not full program generation.

Propose a **hybrid method** that integrates LR parsing with LLMs.

- Refines structural candidates and suggests textual code.
- Examines whether LLMs benefit from LR structural candidates.
- How to choose LR structural candidate **effectively**.
- Emerged as a language-agnostic solution.



Integrating LR-parsing with LLM for Code Completion

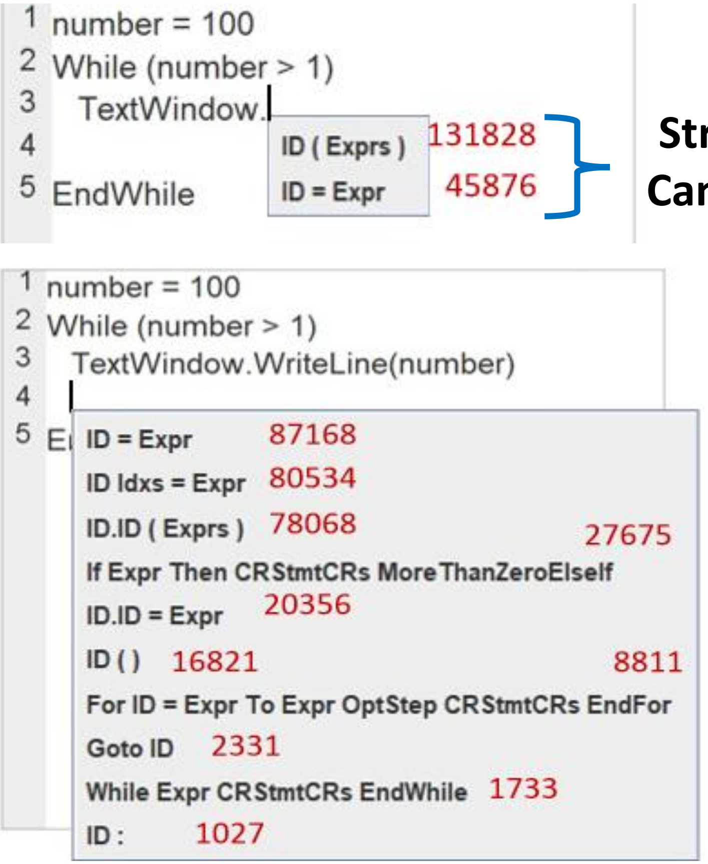
Contributions

Major contributions of our work are as follows:

- Investigates the **maximal improvement** achievable by LLM using ideal LR structural candidates.
- Propose how to **effectively** choose LR structural candidates to **guide LLMs** for better completions.
- **VS Code plugins** for Small Basic and C with **language-agnostic** support.

Our Previous System: Ranked Syntax-Structure Completion

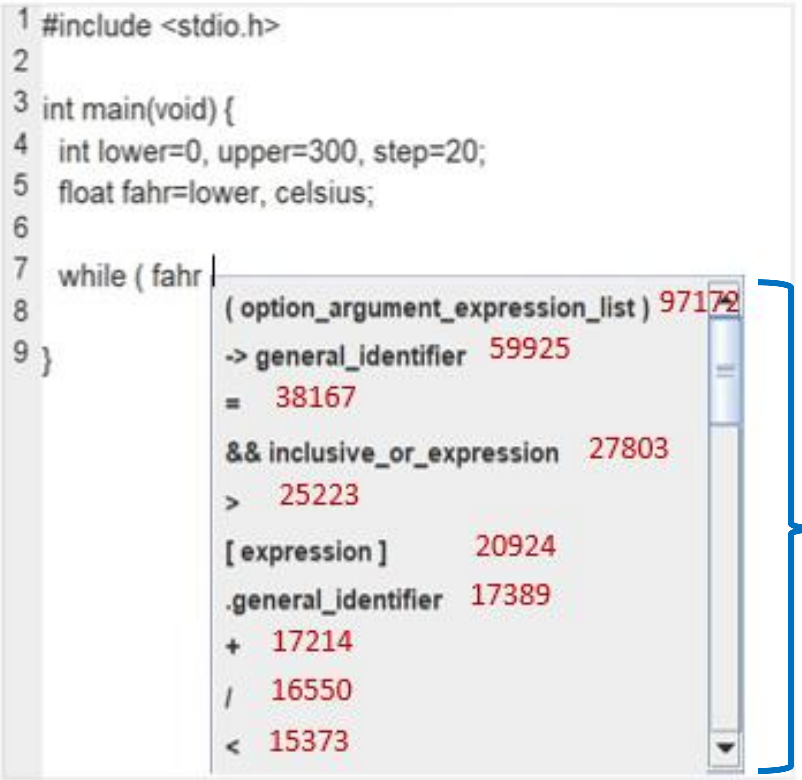
- Suggest only structural candidates.
- Requiring manual editing, time consuming, and uncomfortable.



Structural Candidates

Structural Candidates

Microsoft Small Basic

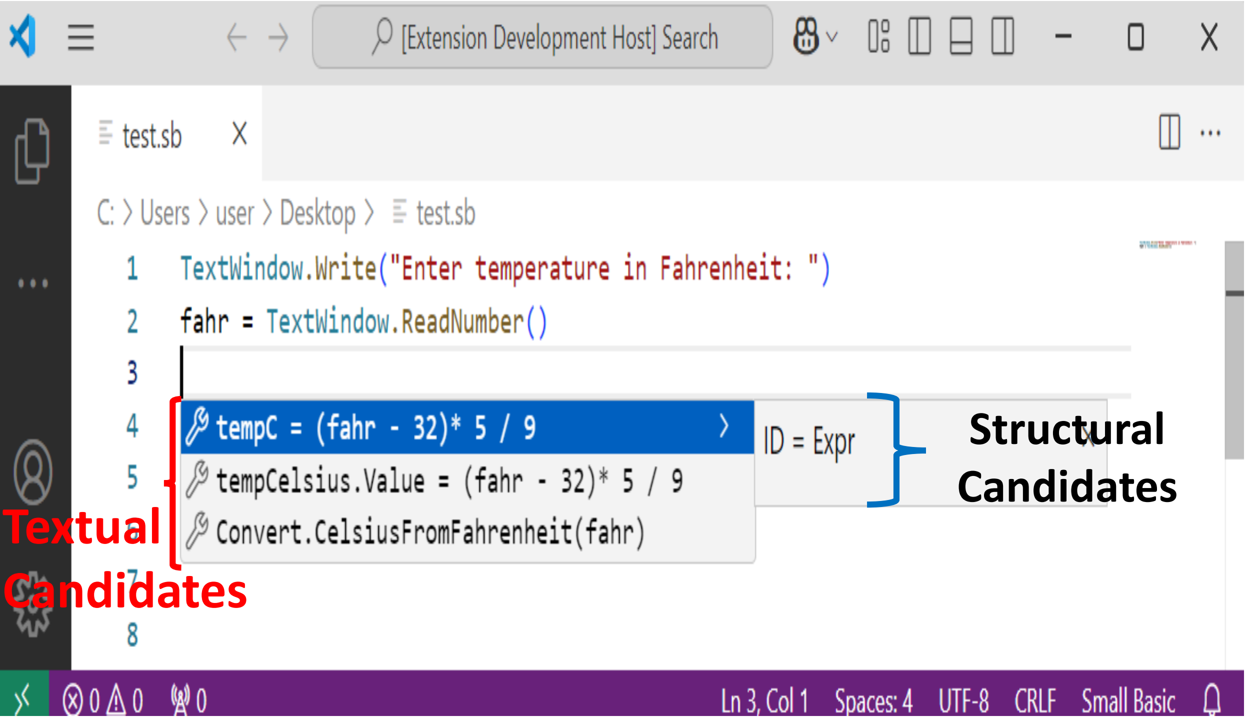


Structural Candidates

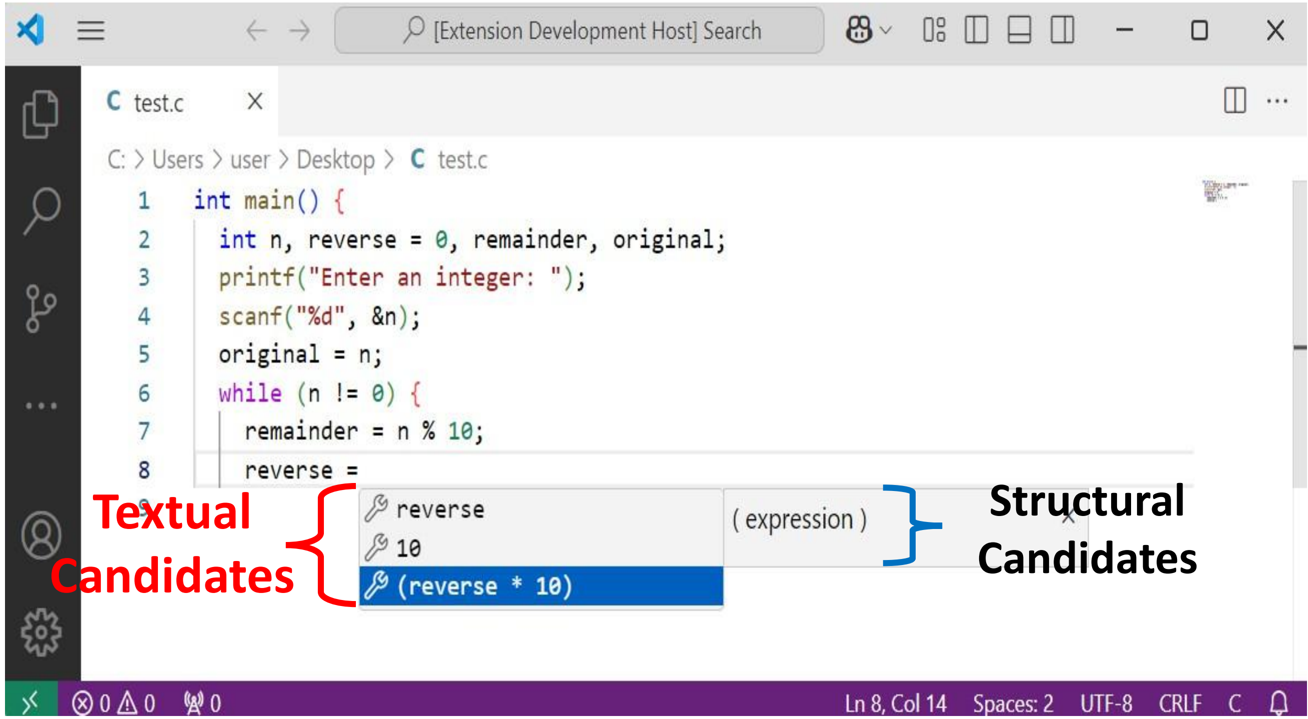
C11

Our Proposed System: LLMs-based Code Completion

- Now suggest textual candidates.
- Enhace usability and productivity.



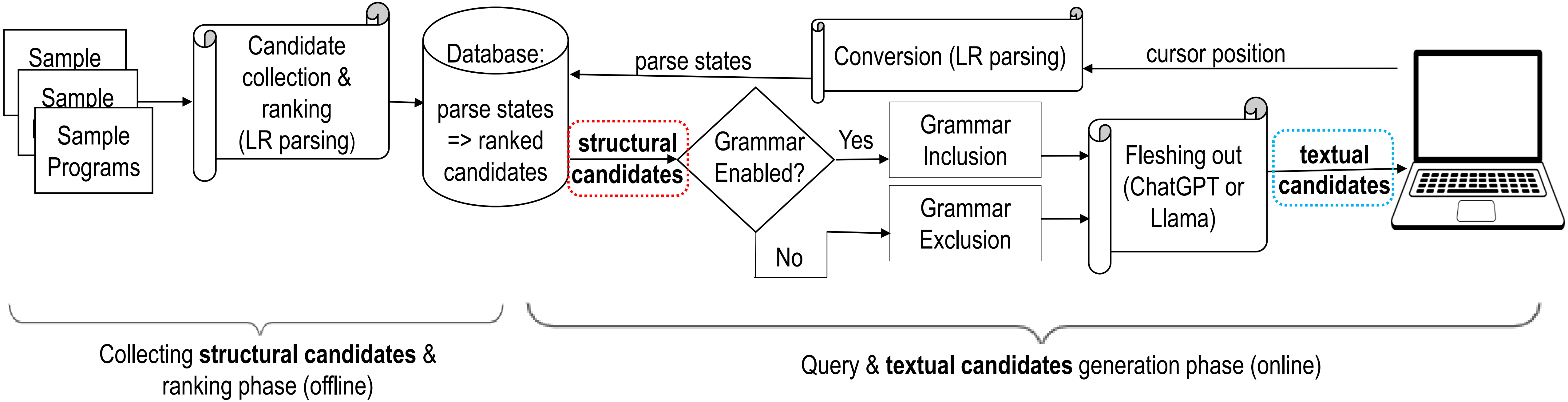
Microsoft Small Basic



C11

Overview of Our System Architecture

Two-phase approach: Collecting & ranking phase and Query phase.



Highlight: The use of ChatGPT or Llama with and without grammar in the system

- It fleshes out the structural candidates to produce textual candidates

- **Ex:** ID.ID(Expr) → TextWindow.WriteLine(“Hello World”)

Offline Phase

Dataset

The training set used for training (candidate collection and built ranked LR structural candidate database):

- SmallBasic community programs: **3,701** programs encompassing nearly 789,023 lines of code.
- C11 open-source projects(cJSON, lcc, bc, gzip, screen, make, tar): **412** programs, totaling approximately 308,599 lines of code.

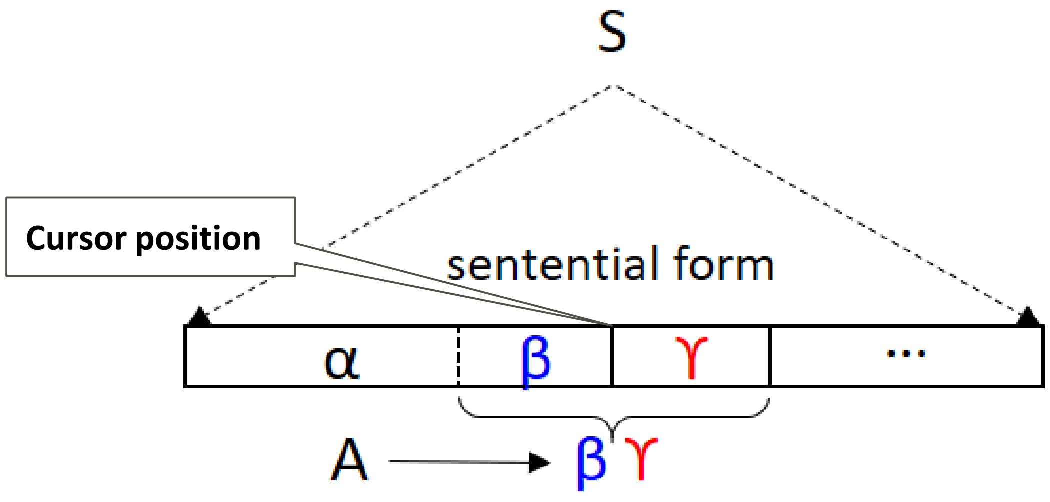
The test set used for evaluation:

- Microsoft SmallBasic tutorial: **27** programs spanning 155 lines of code.
- The Kernighan and Ritchie's book on the C programming language: **106** exercise programs totaling 11,218 lines of code.

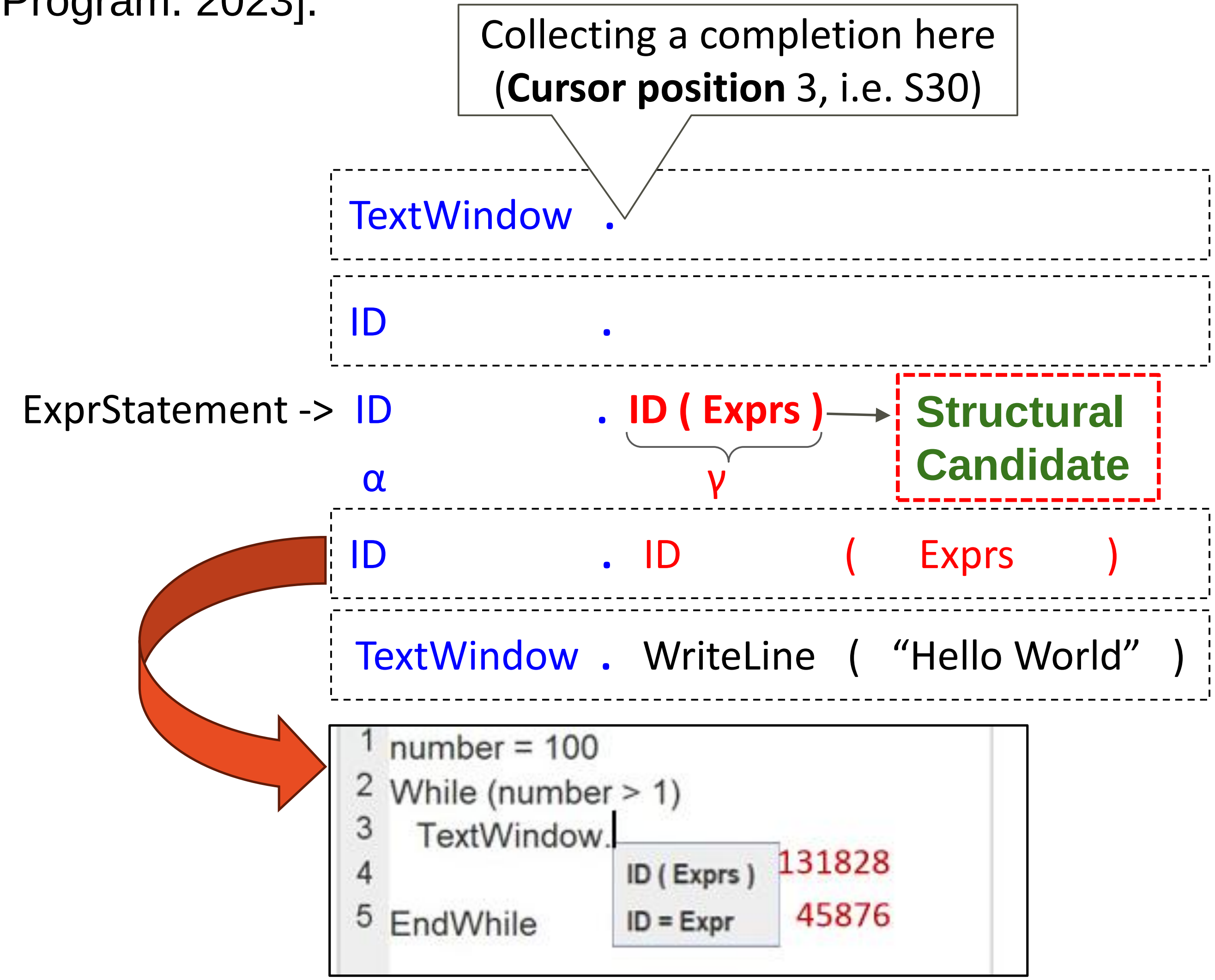
Structural Candidates

What are structural candidates ?

- The concept of suffix sentential form intuitively represents the remaining portion of the program text entered up to the current position [ACM PEPM 2021, Sci. Comput. Program. 2023].



Suffix Sentential Forms



Test Set

A pre-analyzed test set example for Microsoft SmallBasic **'Hello World'** Program, **TextWindow.WriteLine("Hello World").**

Parse State

Line Number , Column Number

0 ID . ID (Exprs) → **Structural Candidate**

1,1: TextWindow
1,11: .
1,12: WriteLine
1,21: (
1,22: "Hello World"
1,35:)

Textual Candidate

6 . ID (Exprs)
1,11: .
1,12: WriteLine
1,21: (
1,22: "Hello World"
1,35:)

30 ID (Exprs)
1,12: WriteLine
1,21: (
1,22: "Hello World"
1,35:)

58 (Exprs)
1,21: (
1,22: "Hello World"
1,35:)

83 STR
1,22: "Hello World"

92)
1,35:)

Online Phase

Structural Candidate Fleshing Out by LLMs

Two crucial research questions in this phase:

- **RQ1.** Do LR structural candidates actually help LLMs?

  If so, how much do they improve code completion?

- **RQ2.** How to choose LR structural candidate effectively?

Prompt Engineering: Prompt Examples for Small Basic

Example of Prompt with Structural Candidate Guidance in Small Basic Language

```
This is the incomplete Microsoft Small Basic programming language code:
number = 100
While (number > 1)
    TextWindow.WriteLine
                  '(Expr)'
Complete the '(Expr)' part of the code in the Microsoft Small Basic
programming language. Just show your answer in place of '(Expr)'.
```

Example of Prompt **without** Structural Candidate Guidance in Small Basic Language

```
This is the incomplete Microsoft Small Basic programming language code:
number = 100
While (number > 1)
    TextWindow.WriteLine
          'next token or line'
Complete the 'next token or line' part of the code in the Microsoft
Small Basic programming language. Just show your answer in
place of 'next token or line'.
```

Prompt Engineering: Prompt Examples for C

Example of Prompt with Structural Candidate Guidance in C Language

```
This is the incomplete C programming language code:
int main(void) {
 char s[1000];
 int i = 0;
 int loop = 1;
            'while (expression) scoped_statement'
Complete the 'while (expression) scoped_statement' part of the code
in the C programming language. Just show your answer in place of
'while (expression) scoped_statement'.
```

Example of Prompt without Structural Candidate Guidance in C Language

```
This is the incomplete C programming language code:
int main(void)
{
   char s[1000];
   int i = 0;
   int loop = 1;
            'next token or line'
Complete the 'next token or line' part of the code in the C programming
language. Just show your answer in place of 'next token or line'.
```

Evaluation Metrics for Code Completion Quality

SacreBLEU Score**:**

- Measures token-level similarity between LLM generated and reference code.
- Formula: $$\mathrm{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

SequenceMatcher Similarity:

- A character-level similarity measure using longest matching subsequences.
- Formula: $$\text{ratio} = \frac{2 \cdot M}{T_A + T_B}$$

RQ1: Evaluation and Results

Evaluation Procedure:

- Automated Prompt Generation: Create prompts using LR structural candidates.
- Code Evaluation: Compare LLM outputs with expected results.
- Systematic Testing: Evaluation across all cursor positions.

Two strategies compared:

  - WithIdealGuide: Prompts include **ideal** LR structural candidate guidance.
  - WithoutGuide: Prompts exclude LR structure candidate guidance.

- **7–14%** precision gain in ChatGPT with LR candidate guidance.

Results:

Table 1: Comparative analysis of experimental results: WithIdealGuide and WithoutGuide.

Programming Language	SacreBLEU (%) With-IdealGuide	SacreBLEU (%) WithoutGuide	SequenceMatcher (%) WithIdealGuide	SequenceMatcher (%) WithoutGuide
Microsoft Small Basic	**49.790**	40.798	**44.703**	37.897
C11	**28.368**	15.472	**28.658**	15.074

RQ1: Precision Comparison in Small Basic and C

- Consistent precision gains across candidate list lengths with ideal structural candidate guidance.

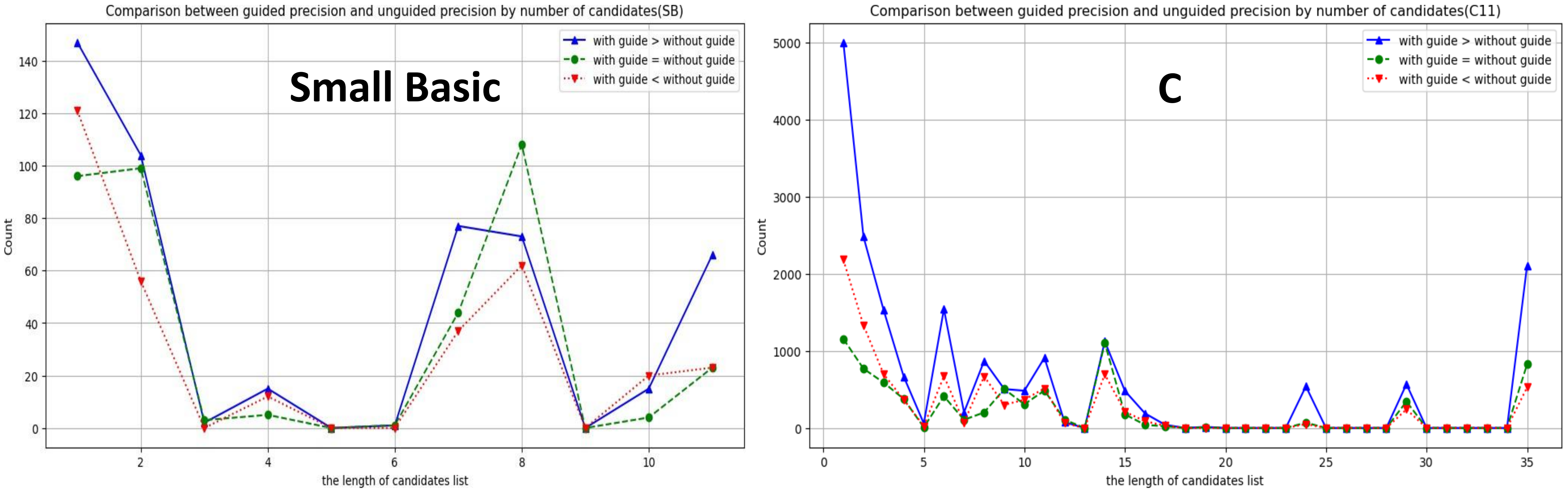


Figure 1: Precision comparison between the results of WithIdealGuide and WithoutGuide.

RQ2: Evaluation and Results

Evaluation Procedure:

- Top 1–3 ranked candidates from prior work often matched correct code.
- Use a pre-ranking database to select top 1–3 candidates for each parser state.

Completion strategies:

1. **WithTop1Guide**: Only the **first ranked structural candidate** is used.

2. **WithinTop3Guide**: The **highest-precision candidate** is selected from the top three ranked completions.

Results:

Table 2: Comparative analysis of experimental results

Programming Languages	Experiment Types	SacreBLEU (%)	SequenceMatcher (%)
Microsoft Small Basic	WithoutGuide	40.798	37.897
	WithIdealGuide	49.790	44.703
	WithinTop3Guide	**45.733**	43.897
	WithTop1Guide	38.524	37.097
C	WithoutGuide	15.472	15.074
	WithIdealGuide	28.368	28.658
	WithinTop3Guide	**26.222**	27.810
	WithTop1Guide	20.217	20.464

RQ2: Precision Comparison in Small Basic and C

- Precision comparison graph showing that WithinTop3Guide outperforms WithoutGuide in SmallBasic.

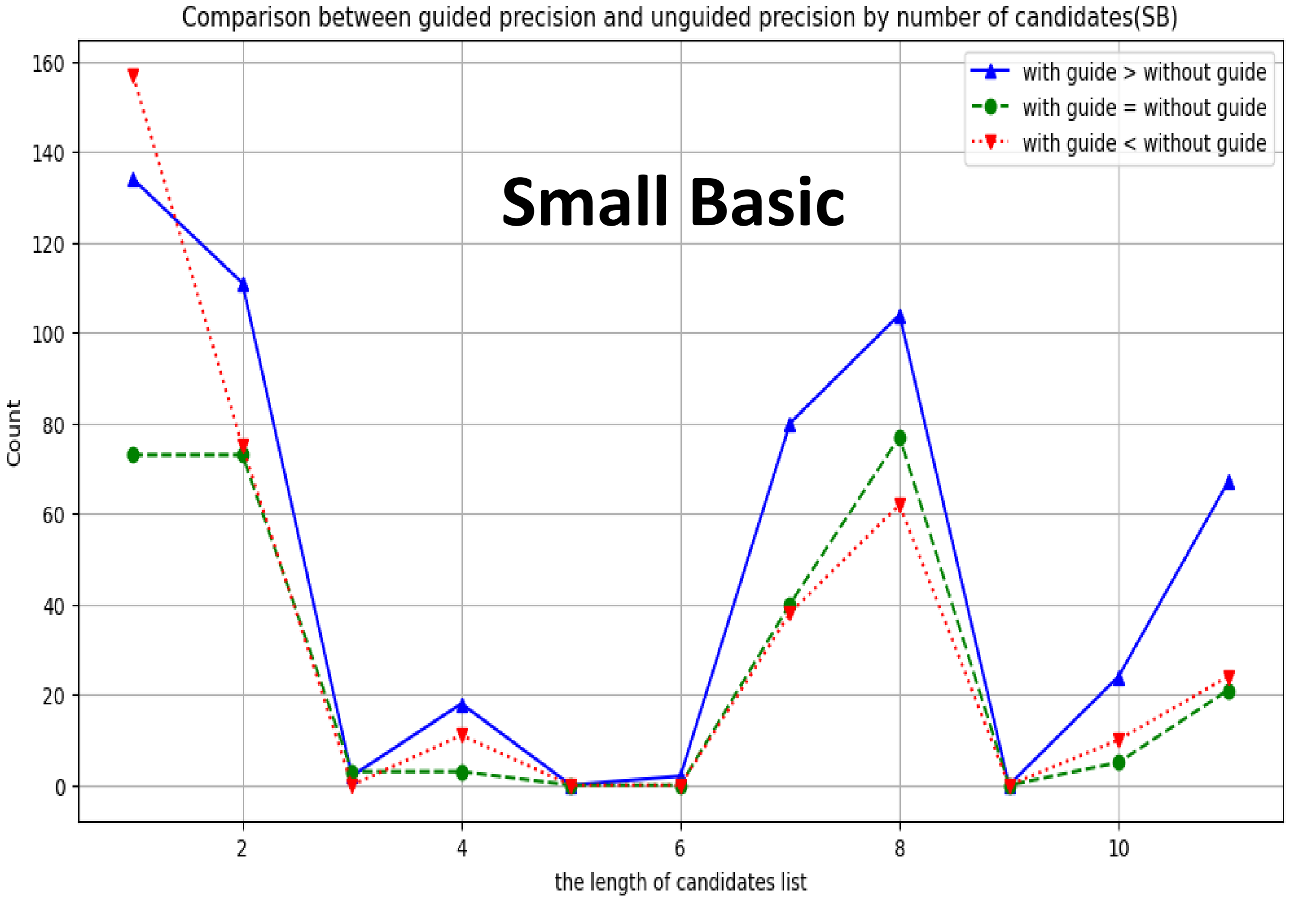


Figure 2: Precision comparison of **WithinTop3Guide** and **WithoutGuide** for different candidate list lengths.

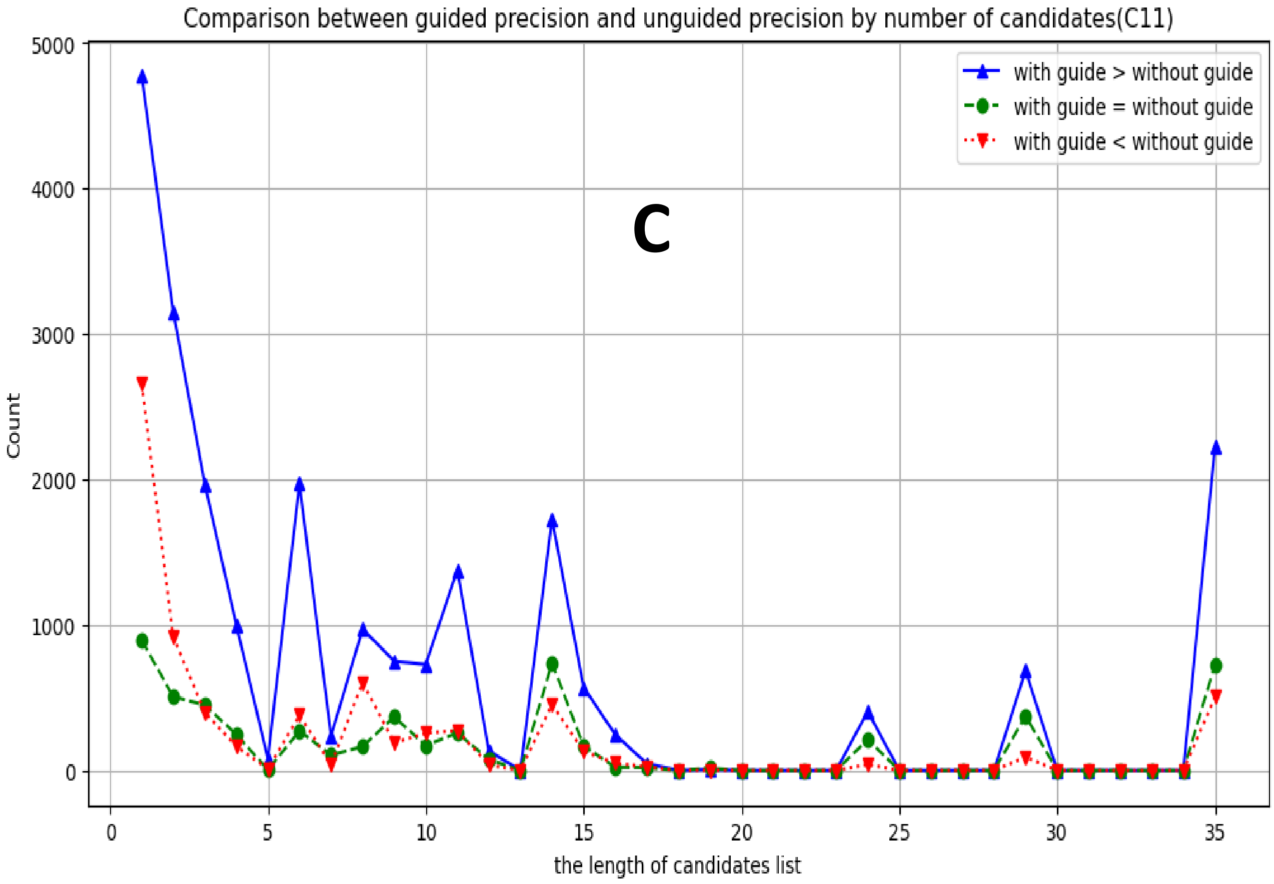


Figure 3: Precision comparison of **WithinTop3Guide** and **WithoutGuide** for different candidate list lengths

Discussion

Discussion: Case Studies on Microsoft Small Basic

- **Best prediction** example in the Microsoft Small Basic experiment.

Parse State: 6 Cursor Position: 5 11
Candidate List: [1: **‘= Expr’**, 2: ‘.ID (Exprs)’, 3: ‘[Expr]’, 4: ‘.ID = Expr’, 5: ‘( )’, 6: ‘[Expr] Idxs’, 7: ‘:’]
Prompt 1: This is the incomplete Microsoft Small Basic programming language code: 2: number = 100 3: While (number > 1) 4: TextWindow.WriteLine(number) 5: number 6: ‘= Expr’ 7: Complete the ‘= Expr’ part of the code in the Microsoft Small Basic 8: programming language. Just show your answer in place of ‘= Expr’.
ChatGPT’s Response WithTop1Guide   **Actual Candidate** = number / 2   = Expr
Actual Textual Answer = number / 2
Response Evaluation SacreBLEU score: 100.0 SequenceMatcher similarity precision: 96.0

Discussion: Case Studies on Microsoft Small Basic

- **Worst prediction** example in the Microsoft Small Basic experiment.

Parse State: 11 Cursor Position: 3 1
Candidate List: [1: ‘ID = Expr’, 2: ‘ID.ID(Exprs)’, 3: ‘ID.ID = Expr’, 4: ‘Sub ID CRStmtCRs EndSub’, 5: ‘ID()’, 6: ‘ID Idxs=Expr’, 7: ‘If Expr Then CRStmtCRs MoreThanZeroElseIf’, 8: ‘For ID=Expr To Expr OptStep CRStmtCRs EndFor’, 9: **‘While Expr CRStmtCRs EndWhile’**, 10: ‘ID:’, 11: ‘Goto ID’]
Prompt 1: This is the incomplete Microsoft Small Basic programming language code: 2: number = 100 3: **‘ID = Expr’** 4: Complete the ‘ID = Expr’ part of the code in the Microsoft Small Basic 5: programming language. Just show your answer in place of ‘ID = Expr’.
ChatGPT’s Response WithTop1Guide number = 100 ID = number * 5 — Actual Candidate While Expr CRStmtCRs EndWhile

Actual Textual Answer
While (number > 1)
  TextWindow . WriteLine(number)
  number = number / 2
EndWhile

Response Evaluation SacreBLEU score: 37.5 SequenceMatcher similarity precision: 33.0

Discussion: Case Studies on C

- **Best prediction** example in the C experiment.

Parse State: 429   Cursor Position: 7 31
Candidate List: [1: **‘NAME VARIABLE’**, 2: ‘CONSTANT’, 3: ‘STRING_LITERAL’, 4: ‘(expression)’, 5: ‘(type_name)cast_expression’, 6: ‘&’, 7: ‘sizeof unary_expression’, 8: ‘sizeof(type_name)’, 9: ‘*’, 10: ‘-’, 11: ‘--unary_expression’, 12: ‘!’, 13: ‘++unary_expression’, 14: ‘ builtin_va_arg(assignment_expression, type_name)’]
Prompt 1: This is the incomplete C11 programming language code: 2: int main(void) 3: { 4: int x = 2, y = 3; 5: printf("x: %d, y: %d\n", x, y); 6: int temp; temp = x; x = y; y = temp; 7: printf("x: %d, y: %d\n", x, 8: ‘NAME VARIABLE’ 9: Complete the ‘NAME VARIABLE’ part of the code in the C11 programming 10: language. Just show your answer in place of ‘NAME VARIABLE’.
ChatGPT’s Response WithTop1Guide: y   **Ideal (Actual) Candidate**: NAME VARIABLE
Actual Textual Answer y
Response Evaluation SacreBLEU score: 100.0 SequenceMatcher similarity precision: 1.00

Discussion: Case Studies on C

- **Worst prediction** example in the C experiment.

Parse State: 246 Cursor Position: 9 36

Candidate List: [1: ‘CONSTANT’, 2: ‘(expression)’, 3: ‘NAME VARIABLE’, 4: ‘sizeof unary_expression’, 5: ‘sizeof (type_name)’, 6: **‘(type_name) cast_expression’**, 7: ‘*’, 8: ‘!’, 9: ‘++ unary_expression’, 10: ‘&’]

Prompt

```
1:  This is the incomplete C11 programming language code:
2:  size_t rem = 0;
3:  while (size >= 1024 && div < (sizeof SIZES / sizeof *SIZES)) {
4:          rem = (size % 1024);
5:          div++;
6:          size /= 1024; }
7:   printf("%6.1f%s ", (float)size +
8:                                  'CONSTANT'
9: Complete the 'CONSTANT' part of the code in the C11 programming language. Just show your answer in place of 'CONSTANT'.
```

ChatGPT’s Response WithTop1Guide	**Ideal (Actual) Candidate**
(SIZES[div-1]>1)?SIZES[div-1] :0.1*SIZES[div-1]	(type_name) cast_expression

Actual Textual Answer
(float)rem

Response Evaluation
SacreBLEU score: 7.6
SequenceMatcher similarity precision: 6.0

Discussion: Analysis on Low Precision

- Our system's low precision results analysis

Numerical Inconsistency	
Our system's selected structural candidate	: ID.ID = Expr
System's response with structural candidate	: GraphicsWindow.Height = 150
Actual textual answer	: GraphicsWindow.Height = 600
SacreBLEU's 1-gram precision	: 80.0

Inconsistency in **Quotation Marks**	
Our system's selected structural candidate	: EndFor
System's response with structural candidate	: 'EndFor'
Actual textual answer	: EndFor
SacreBLEU's 1-gram precision	: 0.0

Discussion: Impact of the Single Candidate

- **Single structural candidate** contribution to prediction results in Microsoft Small Basic.

Parse State: 8 Cursor Position: 7 5

Candidate List: [1: **‘ID = Expr To Expr OptStep CRStmtCRs EndFor’**]

Candidate list of only one structural candidate

Prompt

```
1:  This is the incomplete Microsoft Small Basic programming language code:
2:  For i = 1 To 5
3:        TextWindow.Write("User" + i + ", enter name: ")
4:        name[i] = i
5:  EndFor
6:  TextWindow.Write("Hello")
7:  For
8:        ‘ID = Expr To Expr OptStep CRStmtCRs EndFor’
9:  Complete the ‘ID = Expr To Expr OptStep CRStmtCRs EndFor’ part of
10: the code in the Microsoft Small Basic programming language. Just show
11: your answer in place of ‘ID = Expr To Expr OptStep CRStmtCRs EndFor’.
```

System’s Response with Guidance

```
 i = 1 To 5
TextWindow.Write(name[i] + ", ")
Endfor
```

System’s Response without Guidance

```
i = 1
```

Actual Textual Answer

i = 1 To 5 \n TextWindow . Write (name [i] + ", ") \n EndFor

Result Evaluation

SacreBLEU precision with guidance : 100.0

SacreBLEU precision without guidance : 33.34

Discussion: Integrating Grammar and Different LLMs

❑ Grammar incorporation **[KISM 2025]** :

- Investigated explicit grammar-based guidance.
- No statistically significant improvement.

Table 4: Impact of grammar provision on code completion accuracy.

PLs	Experiment Types	SacreBLEU (%) Without Grammar	SacreBLEU (%) With Grammar	SequenceMatcher (%) Without Grammar	SequenceMatcher (%) With Grammar
Microsoft Small Basic	WithIdealGuide	43.856	**49.790**	42.618	44.703
	WithinTop3Guide	44.773	**45.733**	43.532	43.897
	WithTop1Guide	37.905	**38.524**	36.775	37.097
C11	WithIdealGuide	25.173	**28.368**	26.537	28.658
	WithinTop3Guide	27.385	26.222	28.989	27.810
	WithTop1Guide	21.125	20.217	21.547	20.464

❑ Integrate different LLMs **[ASK 2025]** :

- Evaluated ChatGPT and Llama for LR-based code completion.
- ChatGPT outperforms Llama.

Table 5: Experiment results with guidance using different LLMs.

PLs	LLM Types	SacreBLEU (%)	SequenceMatcher (%)
Microsoft Small Basic	ChatGPT	**43.856**	42.618
	Llama 3	29.086	30.374
C11	ChatGPT	**25.173**	26.537
	Llama 3	15.290	16.913

Implementation: VS Code Extension

Implementation: A VSCode Extension Based on Our Approach

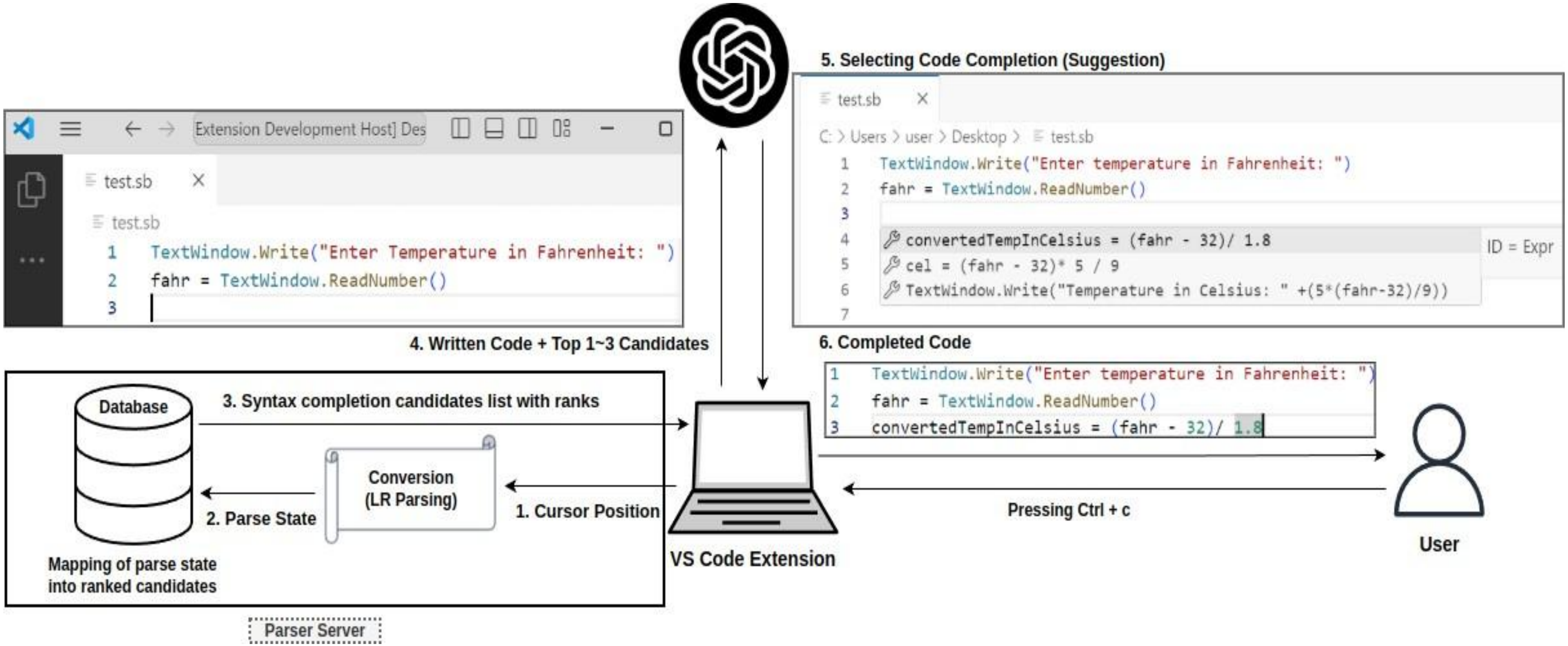
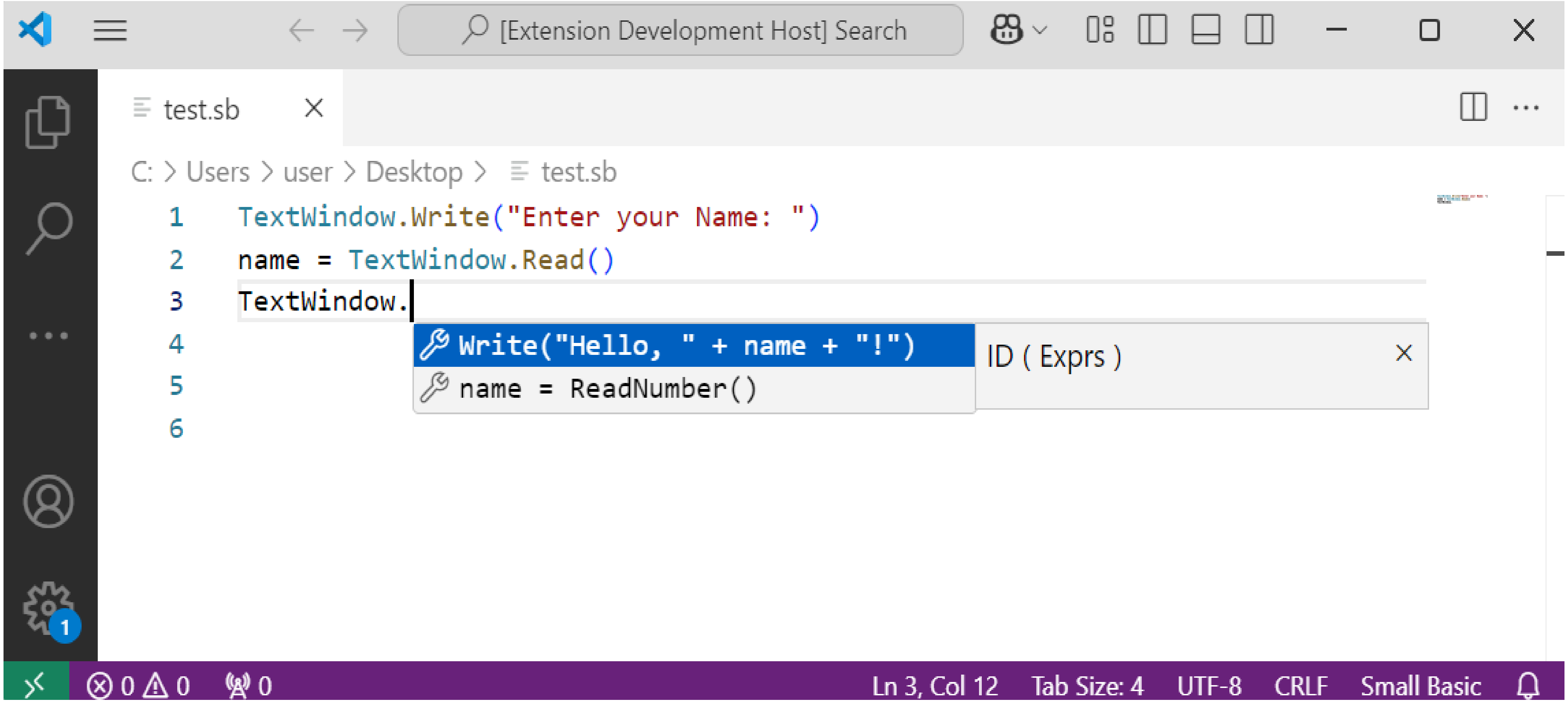


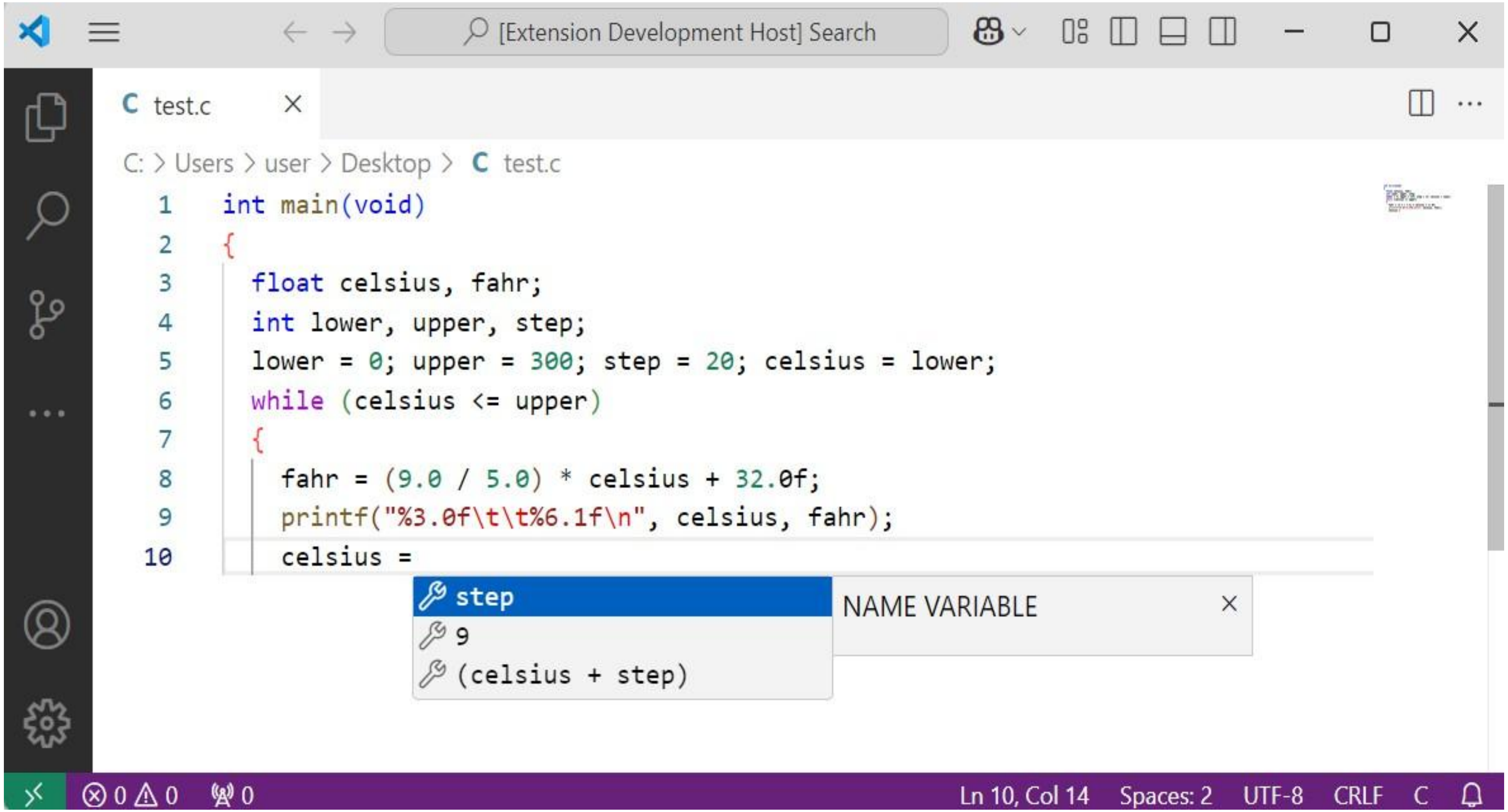
Figure 10: VSCode extension workflow diagram

VS Code Extension: Microsoft Small Basic



Microsoft Small Basic programming VSCode extension example

VS Code Extension: C



C programming VSCode extension example

Related Work

Related Work

❑ LLM-Guided Code Completion Tools:

- **VSCode (IntelliCode Compose)** [1], **JetBrains IntelliJ** [2], and **GitHub Copilot** [3] leverage LLMs for multi-token/multi lines.

  → **Reported Acceptance Rate:** 18% (C) to 38%.
- Our method: fine-grained, controlled LLM-based completion.

  → Achieved **~29%** accuracy on C11 with multi-line completions.
- Recent frameworks (**SynCode**) [4] enforce grammar rules in LLM outputs.

  → Our work systematically evaluates *grammar provision's impact* on completion accuracy.

❑ Parser-Based Techniques:

- Studies use **ANTLR** [5], **GLR** [6], and **LALR parsers** [7] for syntax-aware completion.
- Our method generates ranked symbol sequences with sentential form prefixes.

❑ Candidate Ranking:

- Traditional ranking uses only *frequency counts* (e.g., GitHub data) or *edit history* (e.g., Eclipse) [8].
- Our ranking combines *symbol frequencies* and LLM-based lexeme expansion.

Conclusion

Conclusion & Future Work

Conclusion:

- Examines the upper bound of improvement in LLM-based code completion.
- How to effectively guide LLMs via ranked structures for better completions.
- VS Code plugins for Small Basic and C with language-agnostic design.

Future Directions:

- More diverse languages & datasets to improve accuracy.
- Evaluate real-world programmer experience.
- Generalized plugin for multi-language support.
- Explore better strategies for applying grammar to the LLM.

List of Publications Based on Our Work

The following publications are based on the research presented in this thesis:

1. Md Monir Ahammod Bin Atique, Hyeon-Ah Moon, Isao Sasano, and Kwanghoon Choi. **"Improving LLM-based Code Completion Using LR Parsing",** Journal of Computer Languages, Elsevier, 2025 [Minor Revisions].

2. Md Monir Ahammod Bin Atique, Kwanghoon Choi, Isao Sasano, and Hyeon-Ah Moon. **"Improving LLM-based Code Completion Using LR Parsing-Based Candidates."** In CEUR Workshop Proceedings, vol. 3754, 10th International Symposium on Symbolic Computation in Software Science (SCSS), Japan, 2024. Available at: https://ceur-ws.org/Vol-3754/paper01.pdf.

3. Md Monir Ahammod Bin Atique 와 최광훈. **"LR 파싱을 활용한 LLM 기반 코드 완성에서 문법 제공 비교 분석"**. 2025 한국스마트미디어학회&한국전자거래학회 춘계학술대회, 중앙대학교.

4. Md Monir Ahammod Bin Atique 와 최광훈. **"LR 파싱을 이용한 이용한 LLM 기반 코드 완성에서 ChatGPT 3.5와 Llama 3의 비교 분석"**. 2025년 한국정보처리학회 연례 심포지엄 (ASK 2025), 경북대학교.

Q & A

Thank You for Listening.

Appendix.

References

[1] Svyatkovskiy, A., Deng, S.K., Fu, S., Sundaresan, N., 2020. Intellicode compose: Code generation using transformer, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA. p. 1433–1443. URL: https://doi.org/10.1145/3368089.3417058, doi:10.1145/3368089.3417058.1

[2] Semenkin, Anton, Vitaliy Bibaev, Yaroslav Sokolov, Kirill Krylov, Alexey Kalina, Anna Khannanova, Danila Savenkov et al. "Full Line Code Completion: Bringing AI to Desktop." arXiv preprint arXiv:2405.08704 (2024).

[3] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. “Productivity Assessment of Neural Code Completion”. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (MAPS 2022), pages 21–29. https://doi.org/10.1145/3520312.3534864.

[4] S. Ugare, T. Suresh, H. Kang, S. Misailovic, and G. Singh, "Syncode: LLM generation with grammar augmentation," arXiv preprint arXiv:2403.01632, 2024.

[5] Parr, T., Fisher, K., 2011. LL(*): The foundation of the ANTLR parser generator, in: Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 425–436. doi:10.1145/1993498.1993548.

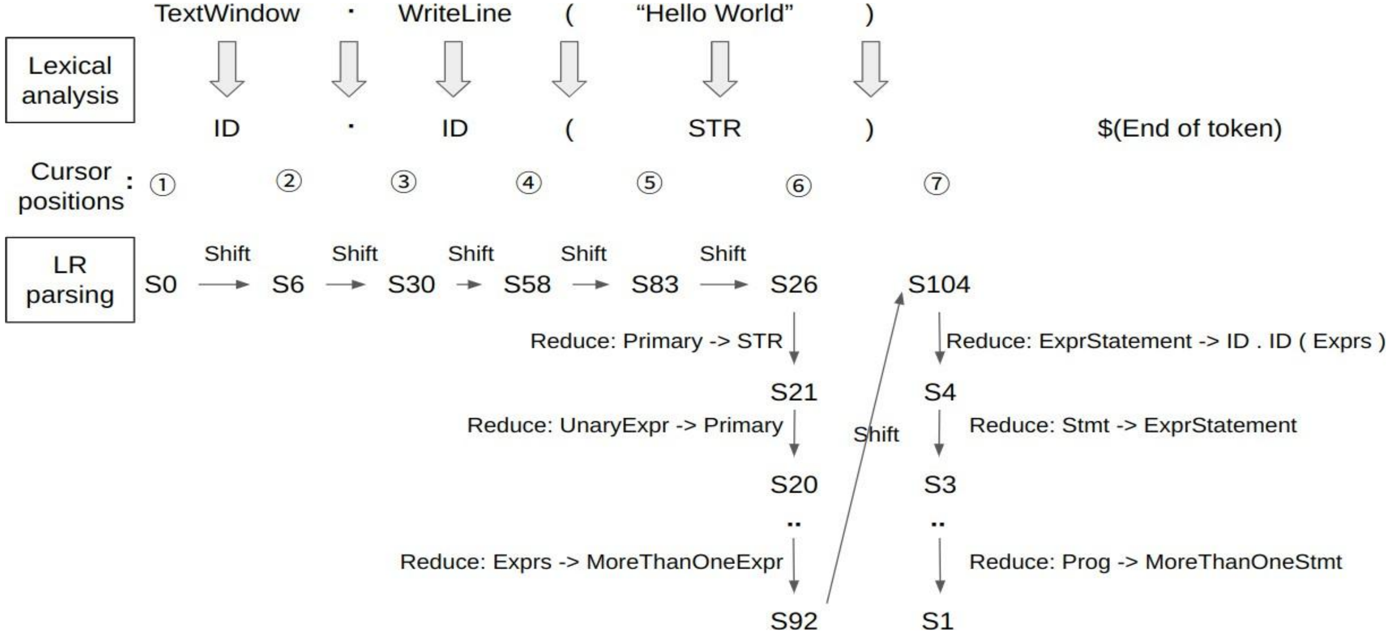
[6] Tomita, M., 1985. Efficient parsing for natural language: A fast algorithm for practical systems. Kluwer Academic Publishers. doi:10.1007/ 978-1-4757-1885-0

[7] Sasano, I., Choi, K., 2023. A text-based syntax completion method using LR parsing and its evaluation. Science of Computer Programming, 102957URL: https://www.sciencedirect.com/science/article/pii/S0167642323000394, doi:https://doi.org/10.1016/ j.scico.2023.102957.

[8] Kersten, M., Murphy, G.C., 2006. Using task context to improve programmer productivity, in: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA. p. 1–11. URL: https://doi.org/10.1145/1181775.1181777, doi:10.1145/1181775.1181777.

LR Parsing in Collecting Structural Candidates

- Collecting candidates using LR parsing (Hello World program in Microsoft Small Basic).
- The left table lists viable prefixes, parser states, predicted candidates, and ranks.
- The right table maps cursor positions to parser states.



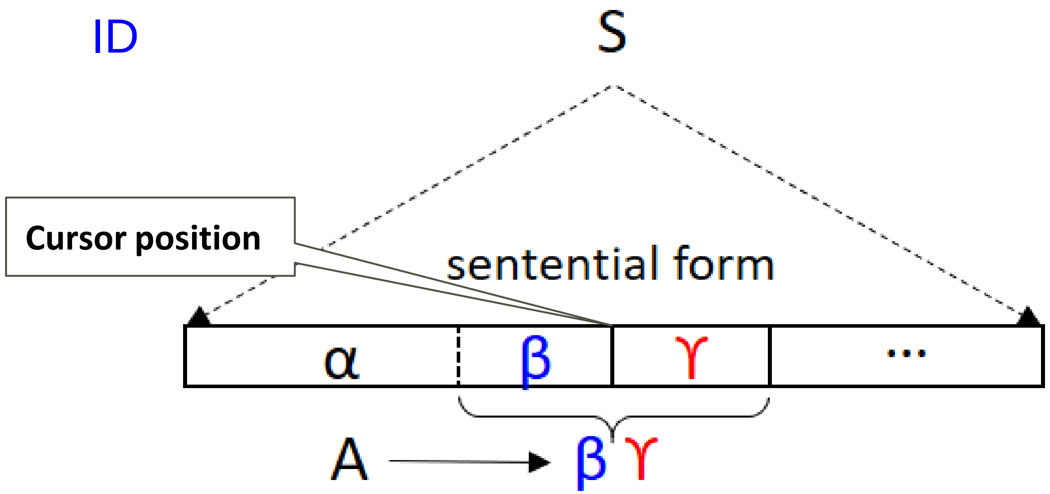
Viable Prefixes	States	Candidates	Ranks
ϵ	S0	ID . ID (Exprs)	1
ID	S6	. ID (Exprs)	1
ID .	S30	ID (Exprs)	1
ID . ID	S58	(Exprs)	1
ID . ID (	S83	STR	1
ID . ID (STR	S26		
ID . ID (Primary	S21		
ID . ID (UnaryExp	S20		
…	…		
ID . ID (Exprs	S92	)	1
ID . ID (Exprs)	S104		
ExprStatement	S4		
Stmt	S3		
…	…		
Prog	S1		

Cursor Position	States
1	**S0**
2	**S6**
3	**S30**
4	**S58**
5	**S83**
6	**S26**
6	**S21**
6	**S20**
6	**…**
6	**S92**
7	**S104**
7	**S4**
7	**S3**
7	**…**
7	**S1**

Structural Candidates Using the Concept of LR Items

- Explains structural candidates using the concept of LR items.
- Every LR item is a dotted production, represented as $A \rightarrow \beta \cdot \gamma$
- If a user writes a text ending with the symbols β preceding the dot and requests completion suggestions afterward, γ can serve as a structural candidate to complete β .
- Consider state S6 with the viable prefix ID in 'Hello World' program.
- TextWindow . WriteLine("Hello World")

ID



Suffix Sentential Forms

01: [Stmt → ID · :, $]

02: [Stmt → ID · :, CR]

03: [ExprStatement → ID · = Expr, $]

04: [ExprStatement → ID · = Expr, CR]

05: [ExprStatement → ID · . ID = Expr, $]

06: [ExprStatement → ID · . ID = Expr, CR]

07: [ExprStatement → ID · . ID (Exprs), $]

08: [ExprStatement → ID · . ID (Exprs), CR]

09: [ExprStatement → ID · ( ), $]

10: [ExprStatement → ID · ( ), CR]

11: [ExprStatement → ID · Idxs = Expr, $]

12: [ExprStatement → ID · Idxs = Expr, CR]

13: [Idxs → · [Expr], =]

14: [Idxs → · [Expr] Idxs, =]

Prompt Engineering: Prompt Templates

Prompt templates featuring LR structural candidate guidance.

Prompt Template with Structural Candidate Guidance
1: This is the incomplete {Name Of Programming Language} code: 2: {Program Prefix} 3: {Suggested Structural Candidate} 4: Complete the {Suggested Structural Candidate} part of the code 5: in the {Name Of Programming Language}. 6: Just show your answer in place of {Suggested Structural Candidate}.

Prompt Template **without** Structural Candidate Guidance
1: This is the incomplete {Name Of Programming Language} code: 2: {Program Prefix} 3: ‘next token or line’ 4: Complete the ‘next token or line’ part of the code 5: in the {Name Of Programming Language}. 6: Just show your answer in place of ‘next token or line’.

Evaluation Metrics for Code Completion Quality

SacreBLEU Score**:**

- Measures token-level similarity between generated and reference code.
- Based on **n-gram precision** (1-gram used in our experiment).
- Calculates how many tokens in the generated output match the reference.
- Includes a **brevity penalty** to penalize overly short outputs.
- Commonly used for evaluating LLM-generated text across systems.
- Where p_n is precision of n-grams, BP is brevity penalty.

Formula:

$$\mathrm{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

SequenceMatcher Similarity

- A character-level similarity measure using longest matching subsequences.
- Implemented via Python's ***difflib.SequenceMatcher*** class.
- Computes a ratio between 0 and 1 indicating how similar two sequences are.
- Captures fine-grained textual alignment, even with minor differences.
- Complements token-level metrics by focusing on overall sequence structure.
- Where M is the number of matching characters, and T_A and T_B are the lengths of two sequences being compared.

Formula:

$$\text{ratio} = \frac{2 \cdot M}{T_A + T_B}$$

Selecting Top 3 Candidates: Evaluation Procedure for RQ2

- Top 1–3 ranked candidates from prior work often matched correct code, averaging 1.8 for SB and 3.15 for C.
- Use a **pre-ranking database** to select top 1–3 candidates for each parser state.
- Candidates ranked by **frequency of correctness** in previous data.
- **Four completion strategies:**

1. **WithIdealGuide:** The **ideal LR structural candidate** from the database is provided in the prompt. Average precision is computed across all cursor positions using these ideal candidates.
2. **WithinTop3Guide:** The **highest-precision candidate** is selected from the top three ranked completions for each LR parser state. The average of this highest precision value is then calculated for all possible cursor positions.
3. **WithTop1Guide:** Only the **first ranked structural candidate** is used, and average precision is determined across all LR parser states.
4. **WithoutGuide:** When **no candidate** was provided and the prompt simply instructed **‘next token or line’**.

Evaluation Procedure for Addressing RQ2: Evaluation

Table 2: Comparative analysis of experimental results: WithIdealGuide, WithinTop3Guide, WithTop1Guide, and WithoutGuide.

Programming Languages	Experiment Types	SacreBLEU (%)	SequenceMatcher (%)
Microsoft Small Basic	WithoutGuide	40.798	37.897
	WithIdealGuide	49.790	44.703
	WithinTop3Guide	45.733	43.897
	WithTop1Guide	38.524	37.097
C	WithoutGuide	15.472	15.074
	WithIdealGuide	28.368	28.658
	WithinTop3Guide	26.222	27.810
	WithTop1Guide	20.217	20.464

- The precision gap between *WithIdealGuide* and *WithinTop3Guide* is much smaller than that between *WithIdealGuide* and *WithoutGuide*, showing the effectiveness of the top-3 strategy.
- *WithinTop3Guide* achieves **2 to 6 times closer SacreBLEU precision** to *WithIdealGuide* compared to *WithoutGuide*, for both Small Basic and C11.
- Selecting and presenting the **top 3 structural candidates** is a practical and effective strategy for improving code completion precision in both language datasets.

Precision Comparison Graph: WithinTop1Guide in SB

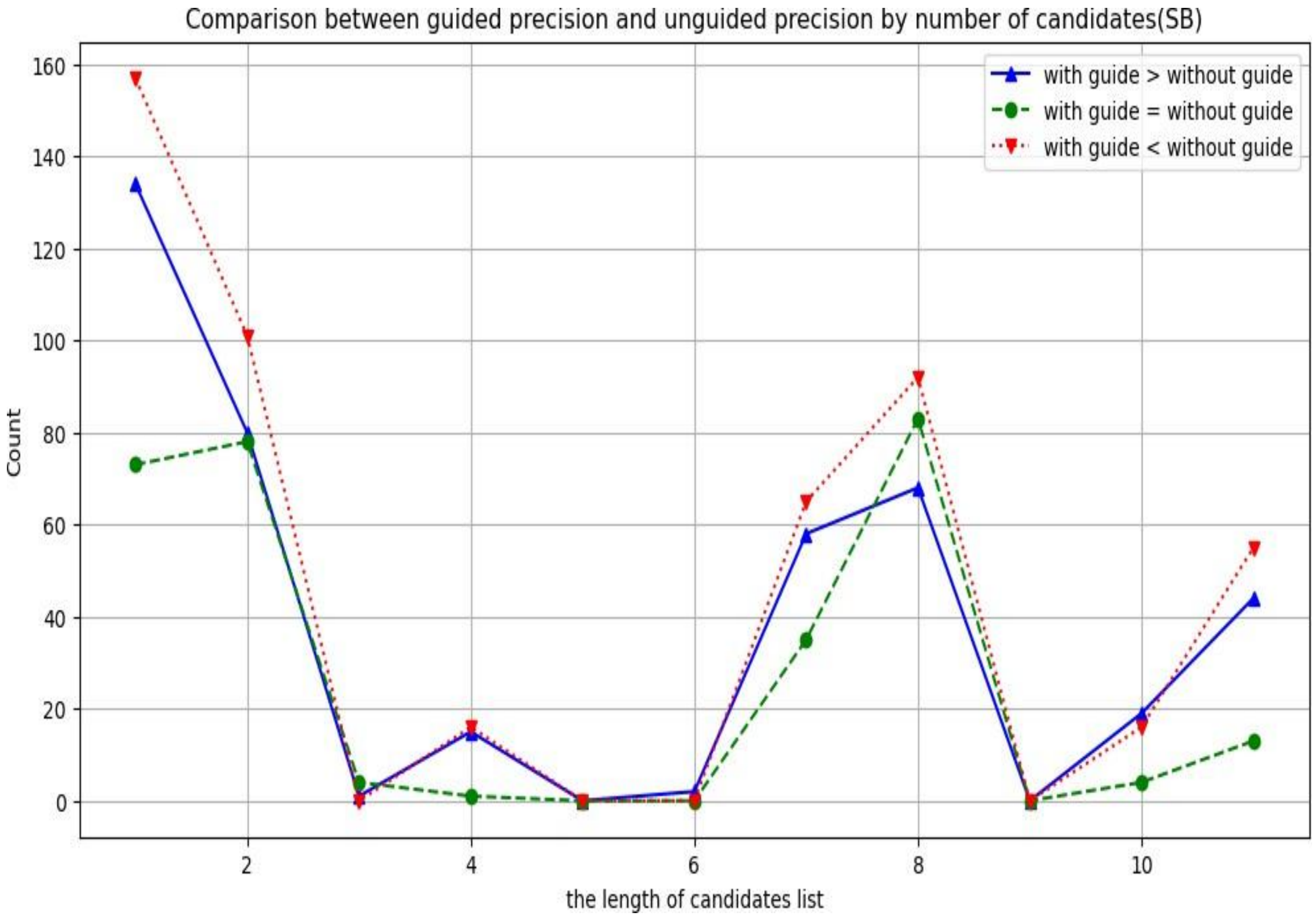


Figure 3: Precision comparison of **WithinTop1Guide** and **WithoutGuide** for different candidate list lengths.

Precision Comparison Graph: WithinTop1Guide in C

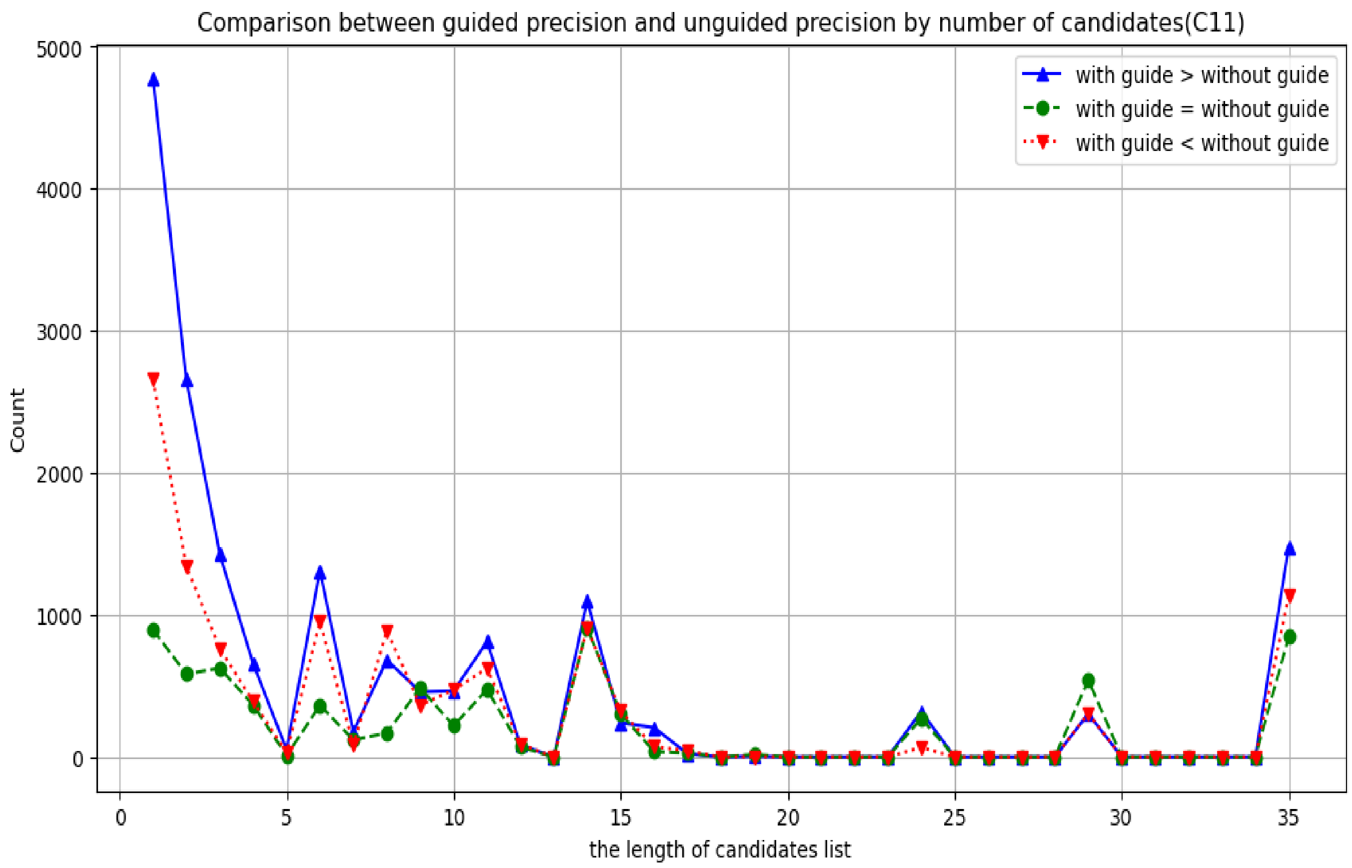


Figure 5: Precision comparison of **WithTop1Guide** and **WithoutGuide** for different candidate list lengths

Analysis of Low Precision

First Example of Numerical Inconsistency	
Our system's selected structural candidate	: MoreThanOneExpr
System's response with structural candidate	: , 10, 50, 100, 150
Actual textual answer	: , 10, 100, 100
SacreBLEU's 1-gram precision	: 62.5
Second Example of Numerical Inconsistency	
Our system's selected structural candidate	: Number
System's response with structural candidate	: 10
Actual textual answer	: 70
SacreBLEU's 1-gram precision	: 0.0

First Example of Inconsistency in Quotation Marks	
Our system's selected structural candidate	:)
System's response with structural candidate	: ')'
Actual textual answer	:)
SacreBLEU's 1-gram precision	: 0.0

Figure 10: Our system's low precision results analysis

Importance of the Top-Ranked Candidate

❑ How often the first suggested candidate was the correct one?

We have since extracted the relevant statistics:

- Under the WithTop1Guide condition, the top-1 structural candidate led to textually correct suggestions in **10.8%** of cases for SmallBasic, and **3.0%** for C11.
- Under the WithTop3Guide condition, the correct textual suggestion appeared within the top 3 in **14.0%** of cases for SmallBasic and **4.4%** for C11.
- When guided by the WithIdealGuide setting, these rates increased significantly: **17.9%** for SmallBasic and **9.7%** for C11.

➢ Higher-quality top-ranked structural candidates lead to better LLM-guided completions.

➢ Current database is limited to a small open-source set.

➢ Expanding to a broader, more diverse corpus is expected to improve both structural and textual precision

Grammar Provision in LLM-based Code Completion using LR parsing

- LLMs improve code completion without explicit grammar knowledge.
- LLM-based code completion uses large training data to generate accurate code without needing explicit syntax rules.
- Due to their probabilistic nature, LLMs can sometimes produce inconsistent or arbitrary outputs.
- Does providing explicit grammar-based guidance improve LLM-based code completion?

➢ A **context-free grammar (CFG)** or **Grammar** defines the syntax of a programming language and structures code completion tasks. CFG includes:

  - **Terminal symbols**: Basic language tokens.
  - **Nonterminal symbols**: Abstract syntactic categories.
  - **Productions**: Rules for forming valid expressions.
  - **Start symbol**: The initial nonterminal for derivations.

➢ Example: *stmt* → ***if*** *(expr) stmt* ***else*** *stmt*
shows terminals (if, else) and nonterminals (stmt, expr).

- In this study, **Small Basic** has 60 production rules; **C language** has 335.

Table 3: Grammatical statistics for Microsoft Small Basic and C

PLs	Microsoft SmallBasic	C11
Num. of prod. rules	61	335
Num. of parse states	119	529
Num. of shift/reduce	816	9209
Num. of goto	222	1907

Prompt Templates with and without Grammar

Prompt template with grammar

1: {Grammar: Production Rules}
2: This is the incomplete {Name of Programming Language} code:
3: {Program Prefix}
4: {Suggested Structural Candidate}
5: Complete the {Suggested Structural Candidate} part of the code
6: in the {Name of Programming Language}.
7: Just show your answer in place of {Suggested Structural Candidate}.

Prompt template without grammar

1: This is the incomplete {Name of Programming Language} code:
2: {Program Prefix}
3: {Suggested Structural Candidate}
4: Complete the {Suggested Structural Candidate} part of the code
5: in the {Name of Programming Language}.
6: Just show your answer in place of {Suggested Structural Candidate}.

Example Program of Grammar Provision in Small Basic

Example of Prompt with Grammar-based Structural Candidate Guidance in Microsoft Small Basic Language:

```
{1: Prog -> MoreThanOneStmt
…. ExprStatement -> ID . ID ( Exprs )
60: Idxs -> [ Expr ] Idxs}
This is the incomplete Microsoft Small Basic programming
language code:
number = 100
While (number > 1)
                    TextWindow . WriteLine
                                '(Expr)'
Complete the '(Expr)' part of the code in the Microsoft Small Basic
programming language. Just show your answer in place of '(Expr)'.
```

System Response: **(number)**

Actual Answer: **(number)**

Example Program of Grammar Provision in C

Example of Prompt with Grammar-based Structural Candidate Guidance in C Language

1: {1: typedef_name -> NAME TYPE
2: ….
3: 335: list_eq1_typedef_declaration_specifier -> declaration_specifier list_eq1_typedef_declaration_specifier}
4: This is the incomplete C programming language code:
5: int main(void)
6: {
7:   char s[1000];
8:    int i = 0;
9: int loop = 1;
10:            ‘while (expression) scoped_statement’
11: Complete the ‘while (expression) scoped_statement’ part of the code
12: in the C programming language. Just show your answer in place of
13: ‘while (expression) scoped_statement’.

System Response: **while (loop) {char s = getchar ();}**

Actual Answer: **while (loop) {char s = getchar ();}**

Overall Evaluation Results of Grammar Provision

Table 4: Impact of grammar provision on code completion accuracy.

PLs	Experiment Types	SacreBLEU (%) Without Grammar	SacreBLEU (%) With Grammar	SequenceMatcher (%) Without Grammar	SequenceMatcher (%) With Grammar
Microsoft Small Basic	WithIdealGuide	43.856	49.790	42.618	44.703
	WithinTop3Guide	44.773	45.733	43.532	43.897
	WithTop1Guide	37.905	38.524	36.775	37.097
C11	WithIdealGuide	25.173	28.368	26.537	28.658
	WithinTop3Guide	27.385	26.222	28.989	27.810
	WithTop1Guide	21.125	20.217	21.547	20.464

Highlights:

- SmallBasic: Grammar helps marginally (e.g., SacreBLEU +6%)
- C11: Mixed results, some worse with grammar
- Overall: No statistically significant improvement (1~6)

Why no major gains?

- LLMs already internalize syntactic knowledge (or grammar) from training data.
- Adding grammar increases prompt length, lead to prompt complexity.
- Grammar constraints can reduce LLM flexibility.

A Comparative Analysis of ChatGPT 3.5 and Llama 3 in Our Work

- Different LLM advancements have greatly improved code completion.
- Compares ChatGPT 3.5 and Llama 3 within an LR parsing-based code completion framework.
- Which model, ChatGPT or Llama, demonstrates better performance in LR parsing-based code generation?
- Used ChatGPT (gpt-3.5-turbo-0125) and Llama 3 (llama-3.1-8b-instant).

Major contribution of this experiment:

- Evaluated LLMs (ChatGPT 3.5 vs. LLaMA 3) for LR-based code completion.
- Model choice significantly affects accuracy.
- ChatGPT outperforms LLaMA in this setting

An Example of Prompt with Different LLMs

Prompt engineering with ideal structural candidate Example of Prompt with Ideal Structural Candidate Guidance in SB

```
1: This is the incomplete Microsoft Small Basic programming
2: language code:
3:  number = 100
4:  While (number > 1)
5:  TextWindow.
6:                    'ID(Expr)'
7: Complete the 'ID(Expr)' part of the code in the Microsoft Small Basic
8: programming language. Just show your answer in place of 'ID(Expr)'.
```

Example of Prompt with Ideal Structural Candidate Guidance in C Language

```
1: This is the incomplete C programming language code:
2: int main(void)
3: {
4:    char s[1000];
5:    int i = 0;
6:    int loop = 1;
7:                   'while (expression) scoped statement'
8: Complete the 'while (expression) scoped_statement' part of the code
9: in the C programming language. Just show your answer in place of
10: 'while (expression) scoped_statement'.
```

Comparative Evaluation of the Example in SB

Comparative Evaluation of the Previous Example in Microsoft Small Basic Language Using ChatGPT 3.5 and Llama 3

ChatGPT 3.5 Response: WriteLine(number)

Response Evaluation:

SacreBLEU (%) score: 100

SequenceMatcher(%) similarity precision: 100

Llama 3 Response: TextWindow.WriteLine

Response Evaluation:

SacreBLEU (%) score: 33.333

SequenceMatcher(%) similarity precision: 43.902

Actual Textual Answer: WriteLine(number

Experimental Results Analysis of ChatGPT and Llama in Our Work

Table 5: Code completion experiment results with ideal structural candidate guidance using different LLMs.

PLs	LLM Types	SacreBLEU (%)	SequenceMatcher (%)
Microsoft Small Basic	ChatGPT	43.856	42.618
	Llama 3	29.086	30.374
C11	ChatGPT	25.173	26.537
	Llama 3	15.290	16.913

Higher accuracy with ChatGPT:

- ChatGPT outperforms Llama 3 with approximately 10–15% higher SacreBLEU and SequenceMatcher scores.
- It produces more precise, structurally aligned completions for both Small Basic and C11.

Significance of model selection:

- Model choice plays a crucial role in syntax-aware code completion.
- ChatGPT integrates structural candidates more effectively than Llama 3.

Database

smallbasic-syntax-completion-candidates

```
State 0
[T, ID, T, =, NT, Expr] : 422
[T, ID, T, ., T, ID, T, =, NT, Expr] : 399
[T, ID, T, ., T, ID, T, (, NT, Exprs, T, )] : 246
[T, ID, T, (, T, )] : 123
[T, ID, T, :] : 59
[T, ID, NT, Idxs, T, =, NT, Expr] : 26
[T, For, T, ID, T, =, NT, Expr, T, To, NT, Expr, NT, OptStep, NT, CRStmtCRs, T, EndFor] : 21
[T, Sub, T, ID, NT, CRStmtCRs, T, EndSub] : 11
[T, While, NT, Expr, NT, CRStmtCRs, T, EndWhile] : 7
[T, Goto, T, ID] : 3
[T, If, NT, Expr, T, Then, NT, CRStmtCRs, NT, MoreThanZeroElseIf] : 1
State 1
State 2
State 3
[T, CR, NT, MoreThanOneStmt] : 207345
State 4
State 5
[T, ID] : 1444
[T, (, NT, Expr, T, )] : 982
[T, STR] : 697
[T, NUM] : 227
[T, ID, NT, Idxs] : 152
[T, ID, T, ., T, ID, T, (, NT, Exprs, T, )] : 121
[T, ID, T, ., T, ID] : 63
```