Beyond Acceptance Rates: The Impact of JetBrains AI Assistant and FLCC

Remco Schrijver

Thesis Committee: A. van Deursen, S. Dumančić, M. Izadi, A. Panichella, P. Derakhshanfar

TU Delft

Index

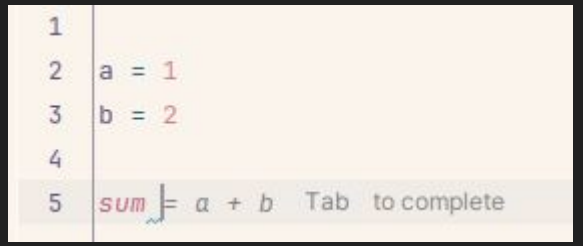
1. Background
2. Research Proposal
3. Dataset
4. Methodology
5. Results & Discussion
    a. User behavior analysis
    b. Code executions
    c. Acceptance rate comparison to benchmarks
    d. Additional findings
6. Concluding remarks

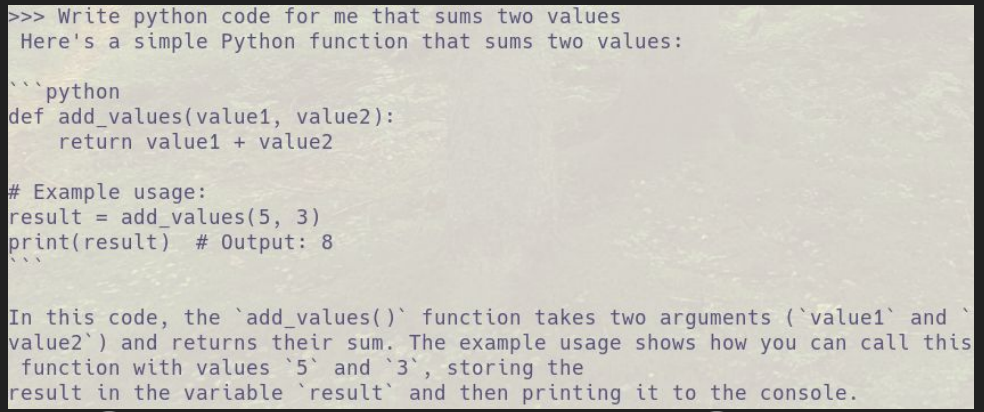
Background

LLM assistance for programming

LLMs, also the technology behind tools like ChatGPT can support users when programming in some of the following ways:

- **Code generation:** User asks in natural language to solve a problem in code
- **Code completion:** The LLM suggests a completion while the user is writing code
- **Test generation:** LLMs generate tests for a given piece of code





JetBrains AI Assistant and FLCC

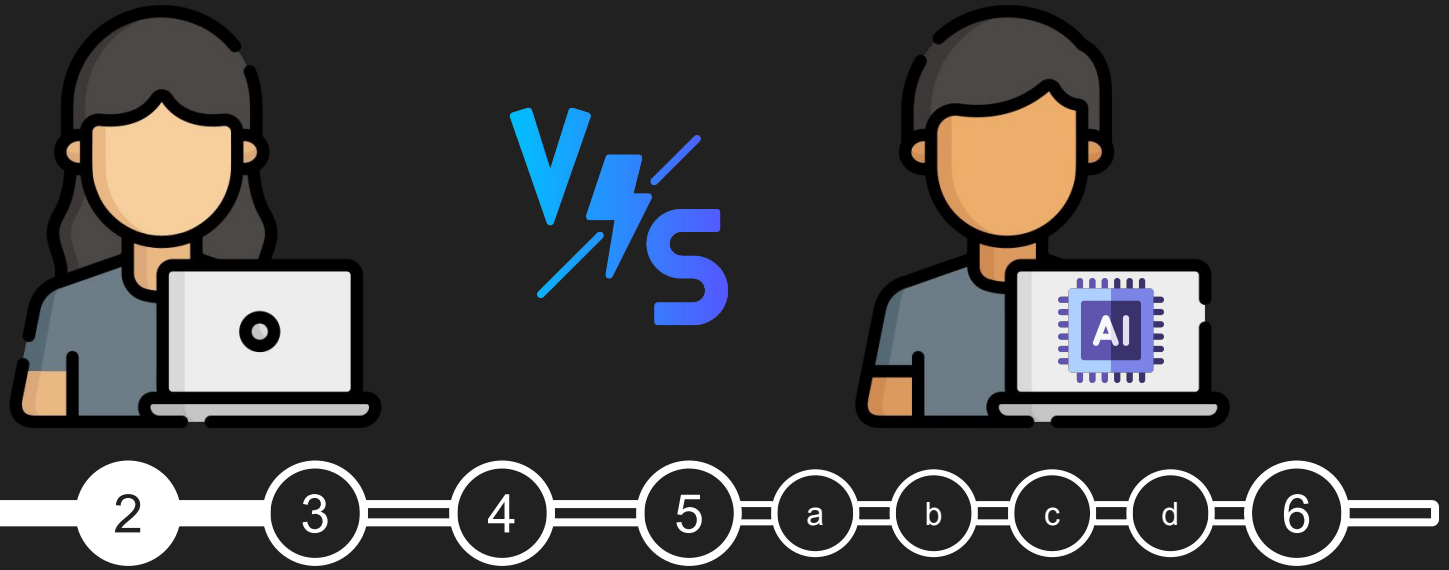
- AI Assistant is backed by OpenAI’s GPT-family of models
  - Can get context from different code files
  - Can do different tasks not only completion
  - Uses an online connection to send requests to the LLM

- FLCC (Full Line Code Completion) is a small GPT/LLaMa-like model
  - Only looks at the context above your caret
  - Only does code completion and only for a single line
  - Runs fully locally on the device of the user

Research proposal

Justification and Gap

Large studies with comparable amounts of data do not focus on comparative analysis and behavior of users with/without LLM support



Dataset

A bit of context on the data

- Sourced from 13 IDEs
- Over 26,000 unique devices/IDE combinations
- Over 43.9 million rows of metrics data
- All data collected from the month of March equally sampled over the days

Differences between IDEs

- Some of the 13 IDEs are very different from each other
- Some are ‘classical’ programming IDEs like IntelliJ (IJ), PyCharm (PC), or CLion (CL)
- Some like DataGrip (DG) and DataSpell (DS) specialize in a task, i.e. database interaction or data science

Columns of interest

- **event_id**, **group_id**, **action_id**, each categorizing the activity this record measures, becomes more precise from left to right.
- **device_id** and **product_code**, allowing us to identify and group records from individual users
- **time_epoch**, allows us to define chronological order of records
- **event_data**, contains more detailed dictionary with data, specific to each type of record, also contains **action_id** for action records.

Methodology

User group creation

From the metric records we found the following user groups:

- AI Assistant Users
- FLCC Users
- Other LLM Users
- FLCC and LLM Users
- Non-LLM or FLCC Users → Our baseline

Results & Discussion

Overview of the results

We will show and discuss the following results:

- User behavior analysis (RQ1)
- Code executions (RQ2)
- Acceptance rate comparison to benchmarks (RQ3)
- Additional findings

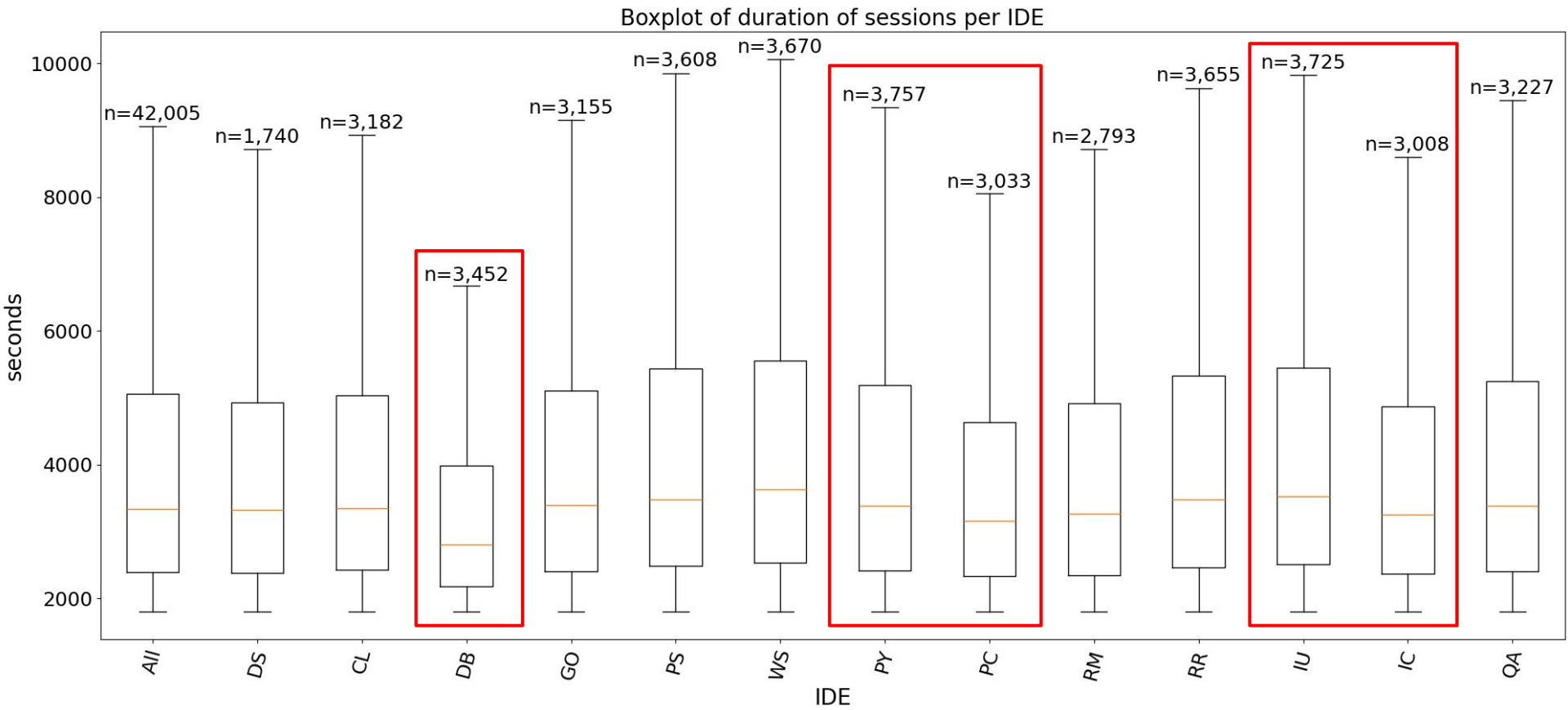
User behavior analysis

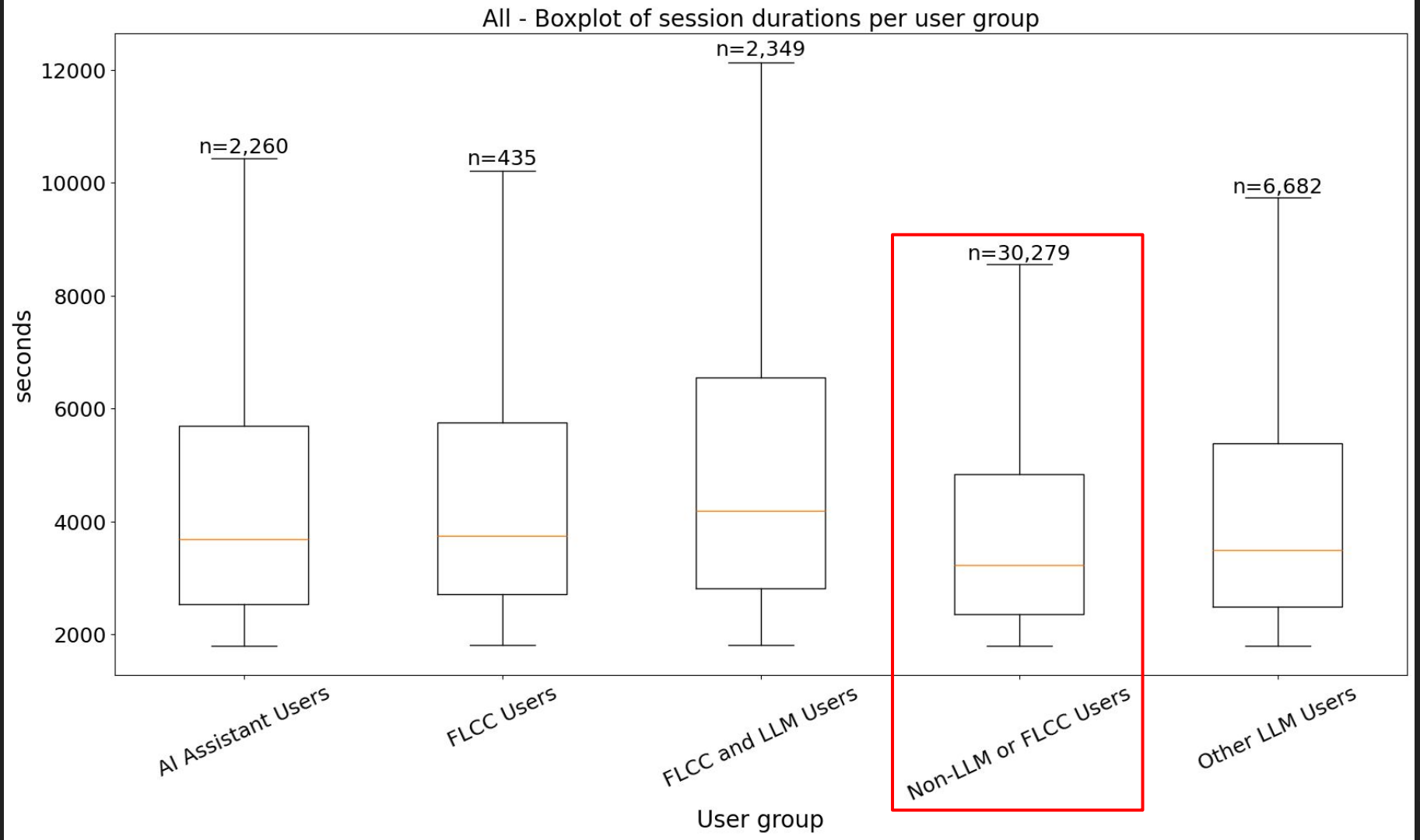
An overview of user behavior results

- Macroanalysis through analyzing actions
  - Session length, typing actions, file switches
  - Compare users with and without LLM support
  - Identify sample sizes to exclude anomalies in discussion

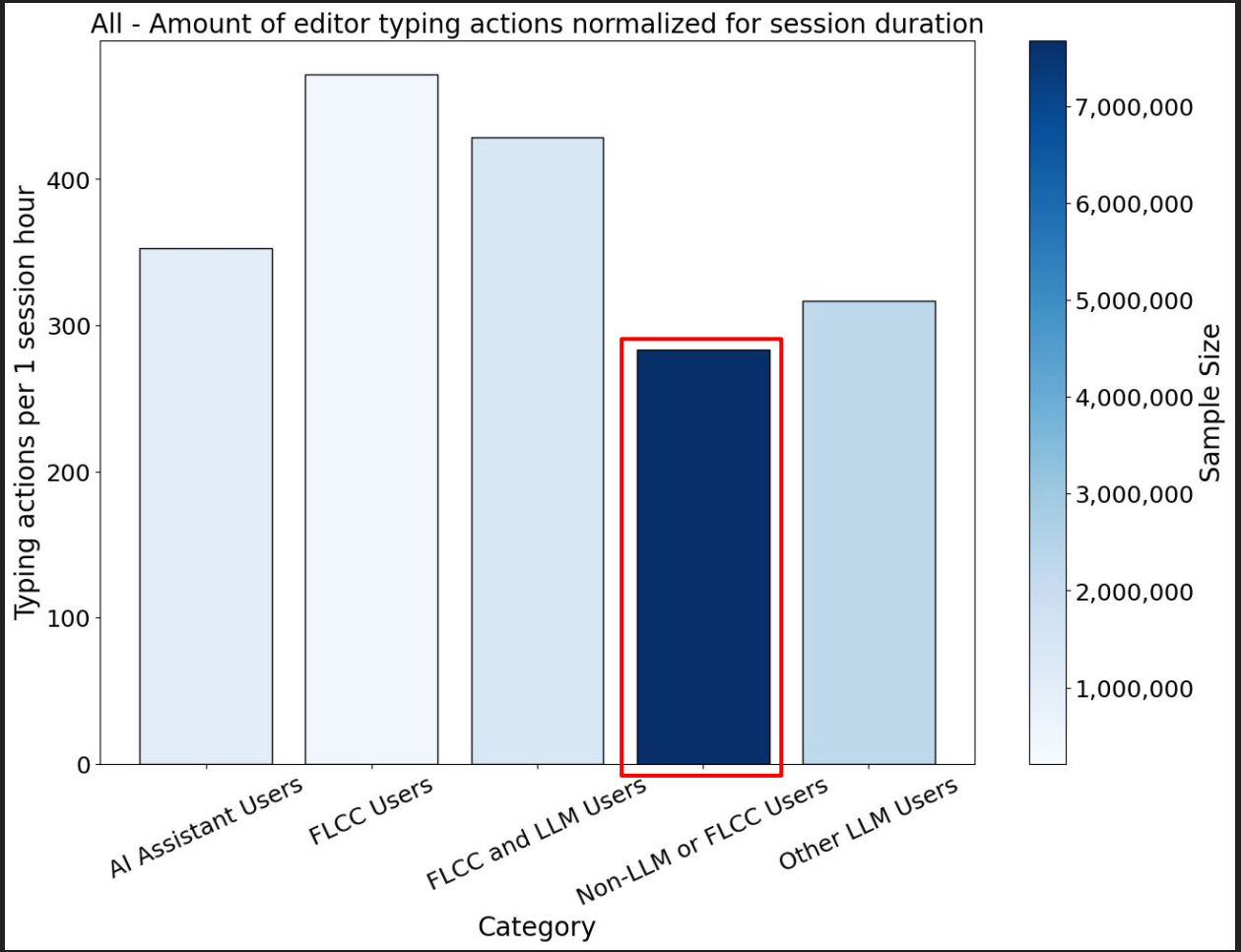
- Microanalysis through n-grams
  - Forced to decrease fidelity to visualize results
  - Hard to do comparative analysis

Macroanalysis

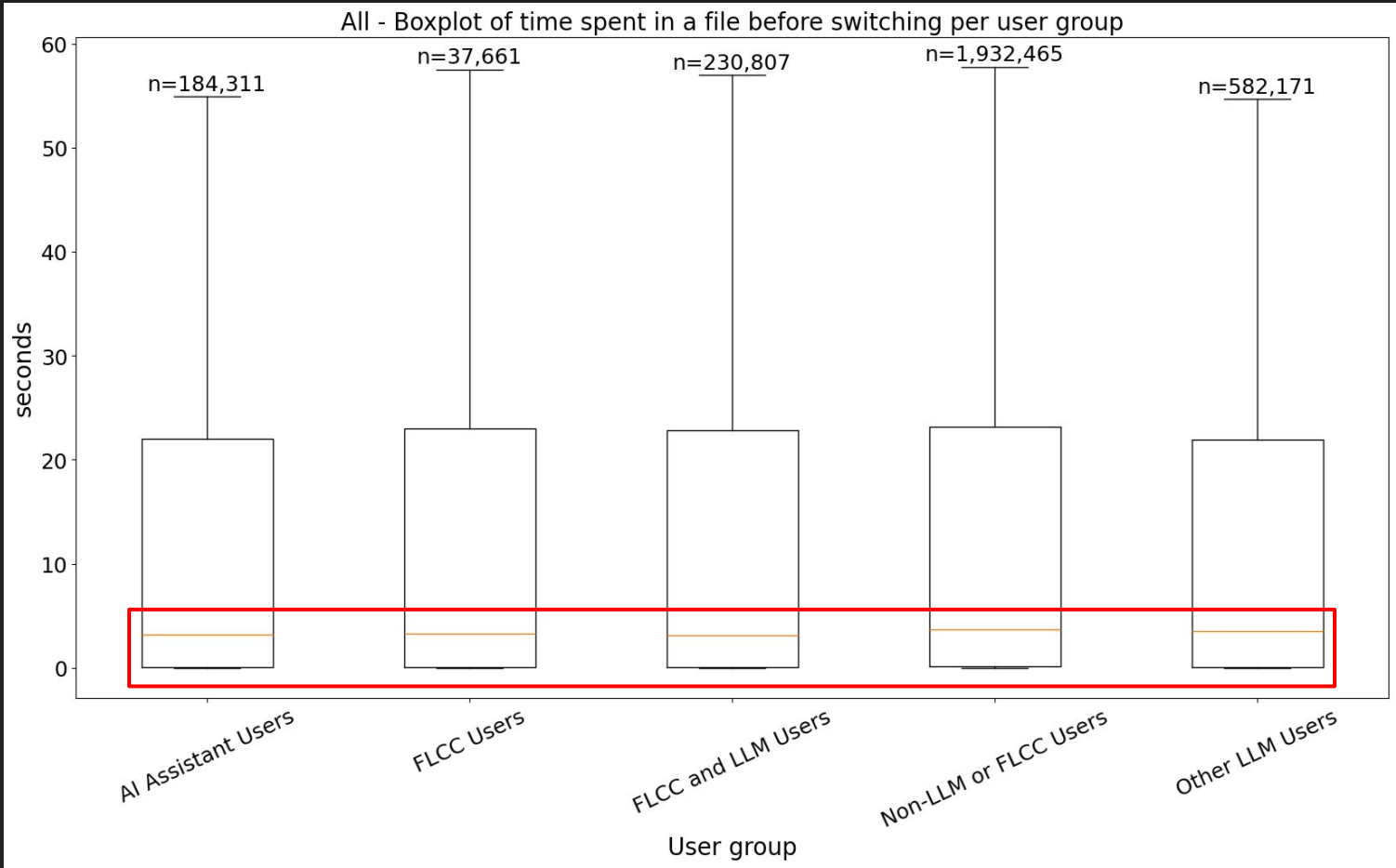

Boxplot of duration of sessions per IDE
seconds
10000
8000
6000
4000
2000
n=42,005
n=1,740
n=3,182
n=3,452
n=3,155
n=3,608
n=3,670
n=3,757
n=3,033
n=2,793
n=3,655
n=3,725
n=3,008
n=3,227
All
DS
CL
DB
GO
PS
WS
PY
PC
RM
RR
IU
IC
QA
IDE


All - Boxplot of session durations per user group
seconds
12000
10000
8000
6000
4000
2000
n=2,260
n=435
n=2,349
n=30,279
n=6,682
AI Assistant Users
FLCC Users
FLCC and LLM Users
Non-LLM or FLCC Users
Other LLM Users
User group

The baseline user group does less typing actions compared to others



Time spent in file is nominally very low.



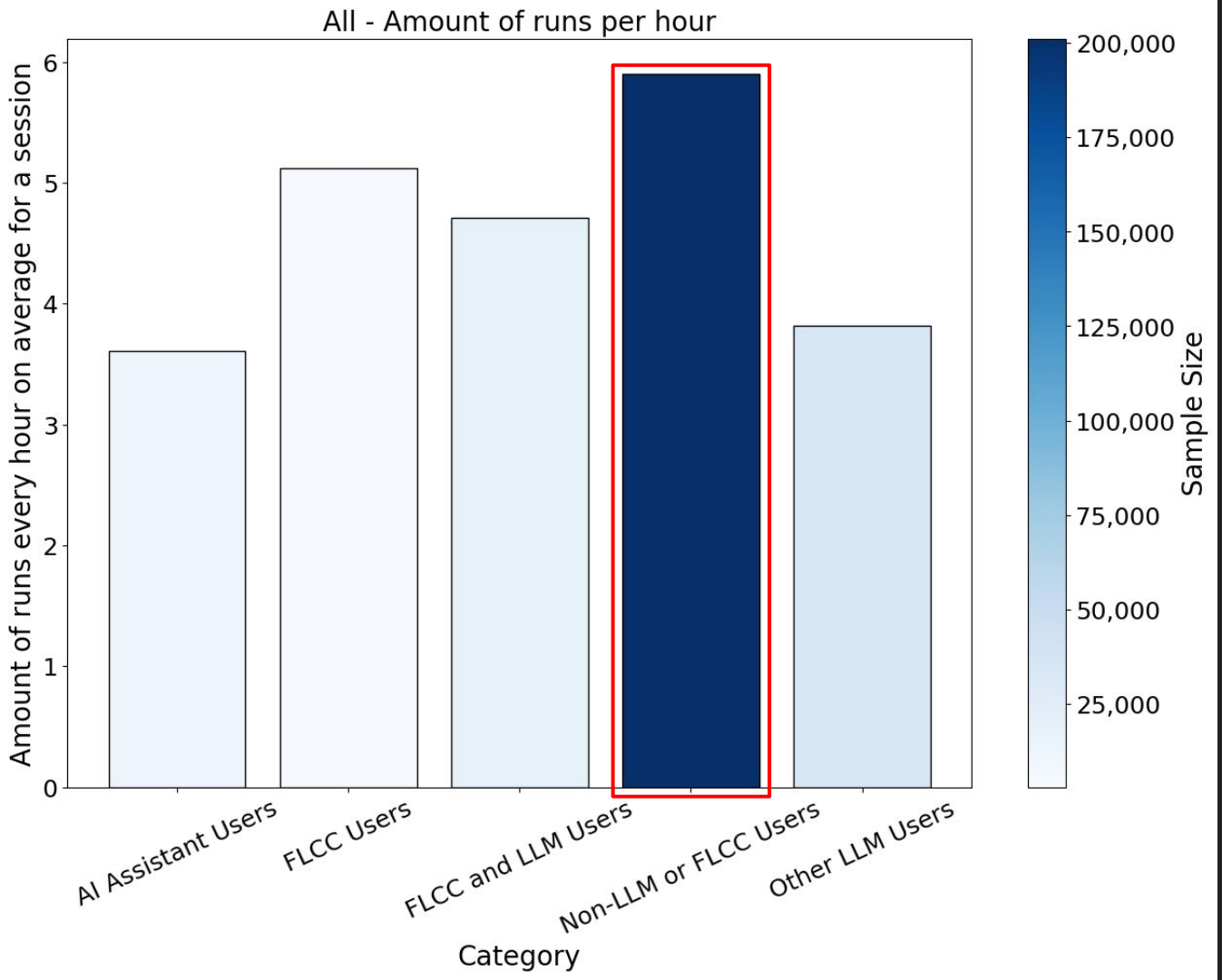
Discussion on user behavior

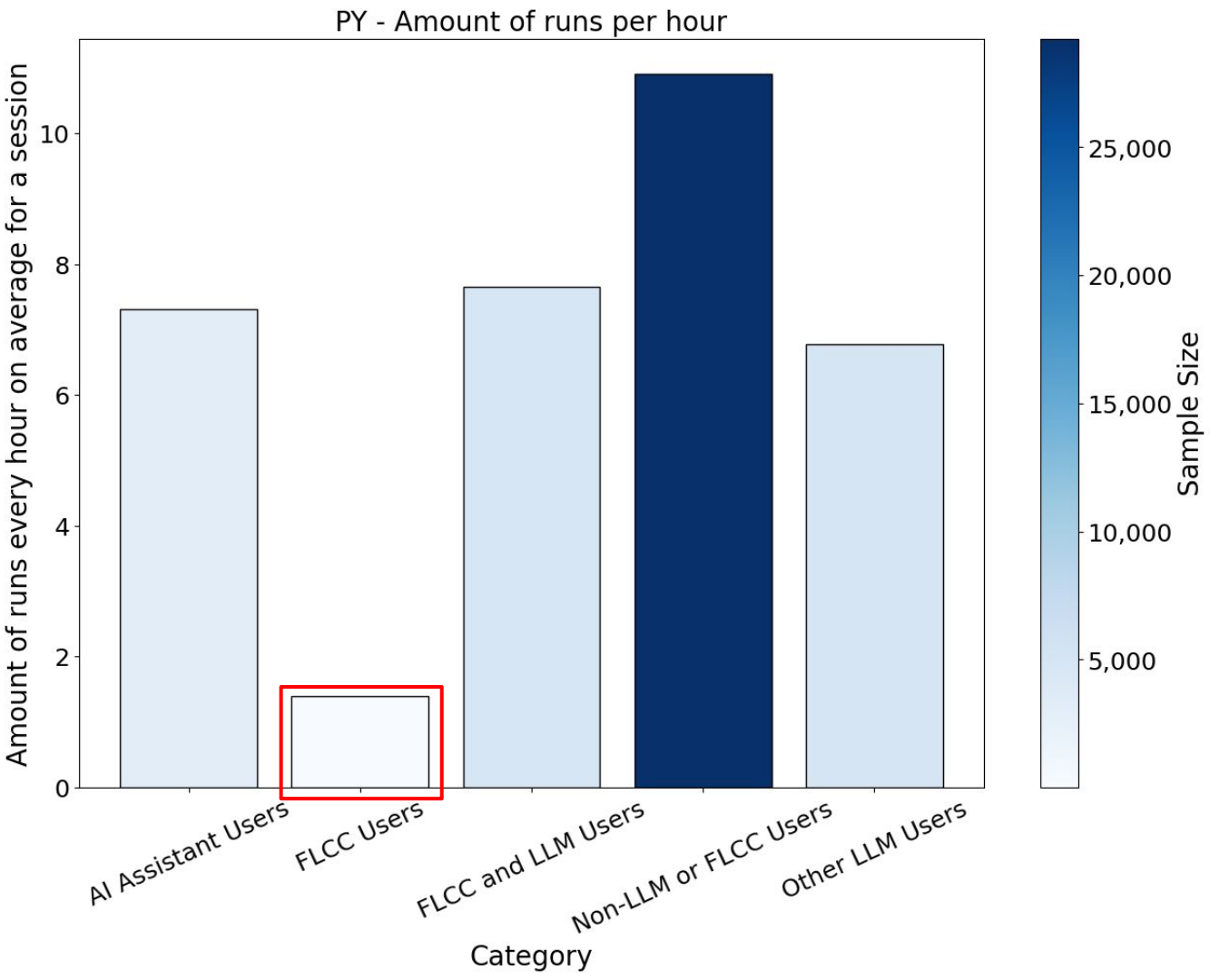
- Increase in session length:
  - Positive: Users stay in a flow state for longer due to reduced frustration and context switching
  - Negative: Users take longer to finish their tasks compared to normal
- More typing actions for LLM supported users:
  - Positive: Users can spend more time typing compared to other activities
  - Negative: Users are forced to do more typing to fix mistakes of LLM provided code
- A mixed result on file switches:
  - Results are mixed per IDE, but general trend is more
  - Results might be flawed considering time spent in a file, which is very low, potentially indicating we included behavior of some IDE tools

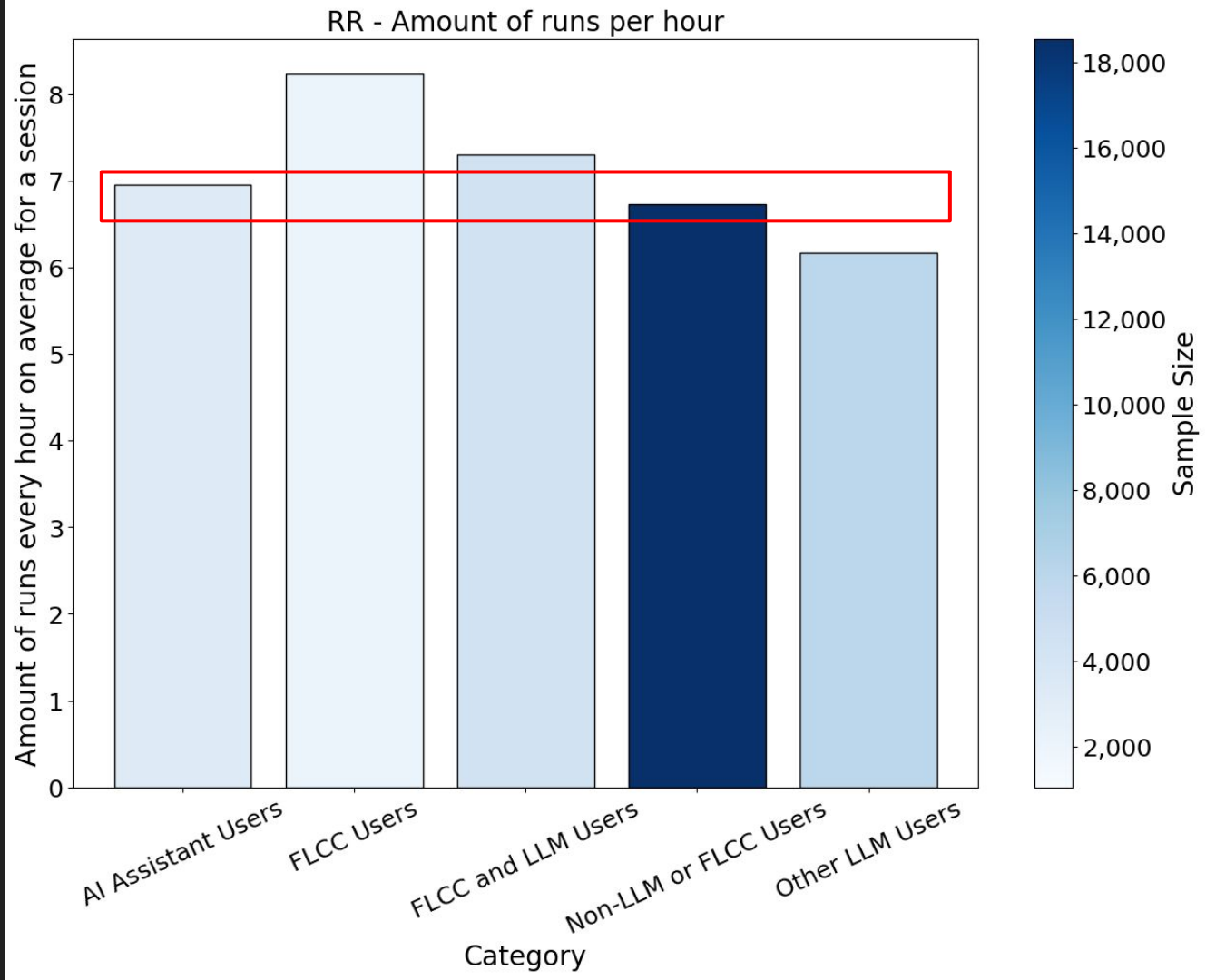
Code Executions

Code executions per session hour

- We use code executions as proxy for (manual) testing
- Code executions not suitable as proxy for each IDE
- General drop in code executions
- We see anomalies when we have low sample sizes
- Some IDEs show increased amount of code executions
- Some IDEs have equivalent amount of code executions


All - Amount of runs per hour
Amount of runs every hour on average for a session
0
1
2
3
4
5
6
AI Assistant Users
FLCC Users
FLCC and LLM Users
Non-LLM or FLCC Users
Other LLM Users
Category
Sample Size
200,000
175,000
150,000
125,000
100,000
75,000
50,000
25,000


PY - Amount of runs per hour
Amount of runs every hour on average for a session
0
2
4
6
8
10
AI Assistant Users
FLCC Users
FLCC and LLM Users
Non-LLM or FLCC Users
Other LLM Users
Category
Sample Size
5,000
10,000
15,000
20,000
25,000


RR - Amount of runs per hour
Amount of runs every hour on average for a session
0
1
2
3
4
5
6
7
8
AI Assistant Users
FLCC Users
FLCC and LLM Users
Non-LLM or FLCC Users
Other LLM Users
Category
Sample Size
2,000
4,000
6,000
8,000
10,000
12,000
14,000
16,000
18,000

Discussion on code executions

The decrease in code executions for particular IDEs could be positive, but based on literature negative scenarios are more likely

- Positive:
  - Users use the LLM suggestions for scaffolding and edit a lot of suggestions, writing critical and more complicated expression themselves
- Negative:
  - Users are slower when completing their tasks with LLMs, therefore a drop in code executions per session hour. This hypothesis is not supported by previous controlled studies
  - Users trust code generated by LLMs more and use them without testing what was written. More likely when looking at existing literature of impact of students supported by LLMs

Acceptance rate comparison to benchmarks

[1]

We use GPT-3.5-Turbo as stand-in for AI Assistant

General pattern for the three language included is this:

- Python and TypeScript have roughly equal results
- Java performs better compared to Python and TypeScript

Table 5.1: Table showcasing benchmark results of GPT-3.5-Turbo as taken from CrossCodeEval [9]. The results included are the languages that we also have acceptance rate data on, so Python, Java, and TypeScript.

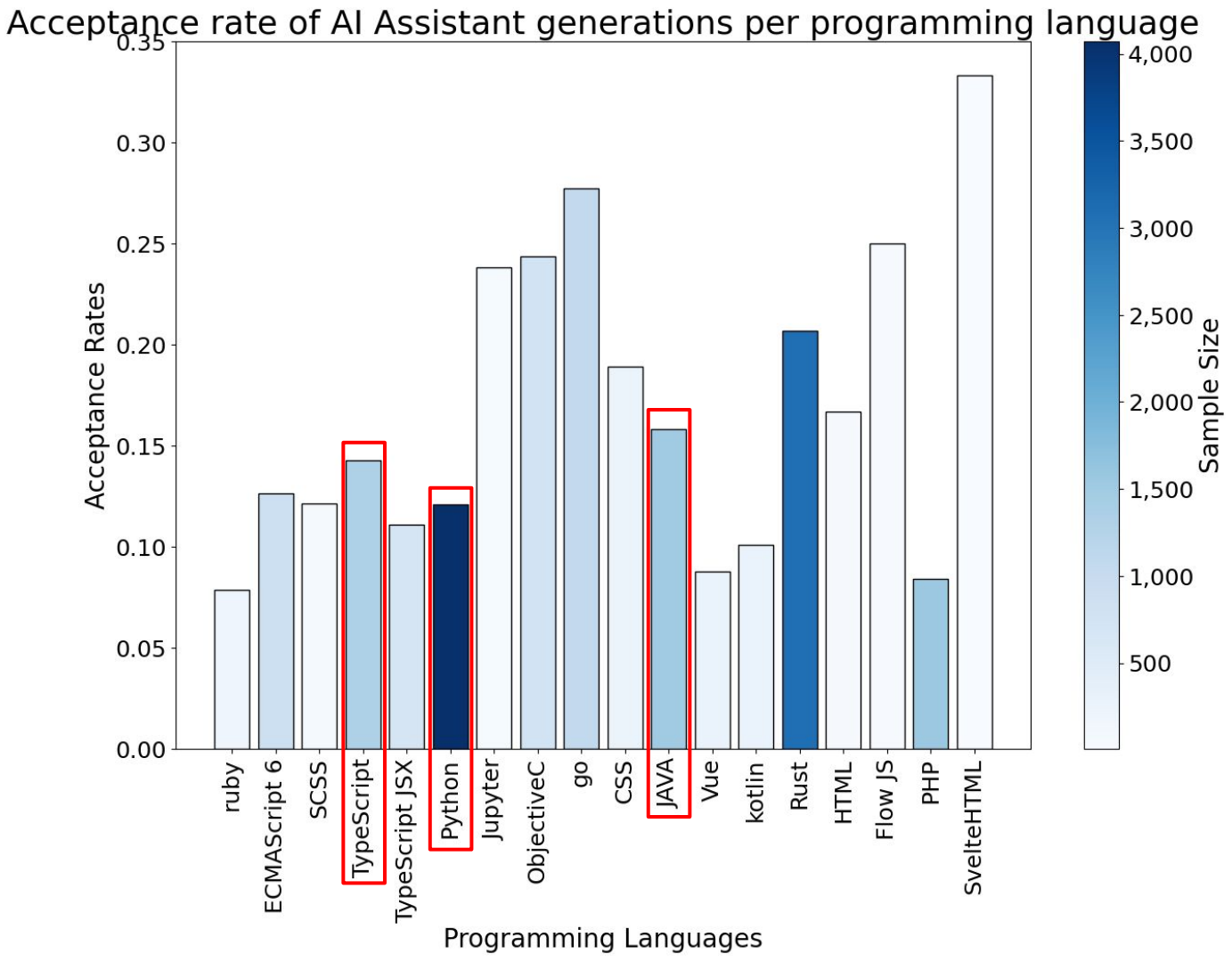
Model for Code Match	Python		Java		TypeScript	
	EM	ES	EM	ES	EM	ES
GPT-3.5-Turbo + Retrieval w/ Ref.	15.72	58.88	22.72	68.50	14.15	58.40
Model for Identifier Match	**Python**		**Java**		**TypeScript**	
	EM	F1	EM	F1	EM	F1
GPT-3.5-Turbo + Retrieval w/ Ref.	23.49	50.14	31.79	60.52	20.65	51.54

[1]: Yangruibo Ding et al. CrossCodeEval: A Diverse and Multilingual Benchmark for Cross-File Code Completion. 2023. arXiv: 2310.11248

The results can be shortened to this:

Java > Python
Java > TypeScript
TypeScript > Python

Results do not align with the results we see in the benchmark.



Discussion on acceptance rates compared to benchmarks

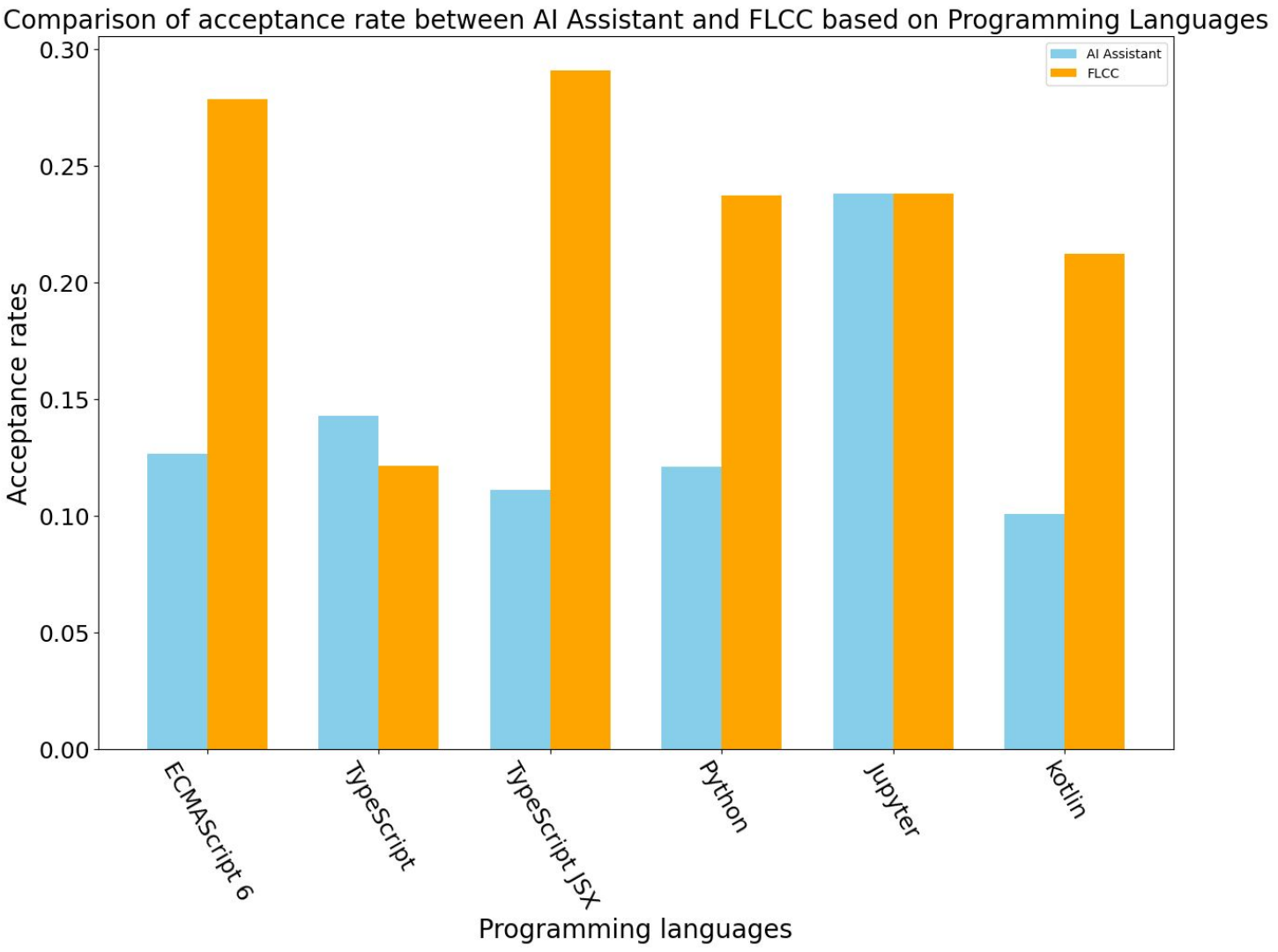
- There seems to be no relationship between acceptance rates of different programming languages and the results on CrossCodeEval
- CrossCodeEval is one of only three benchmarks comparing programming languages, also the most representative of AI Assistant’s process

- Benchmarks do not represent the real-world performance:
  - Metrics used to define benchmark scores could be flawed
- Benchmarks do represent real-world performance:
  - Users accept incorrect results to edit them after accepting

Additional findings

Performance disparity between FLCC and AI Assistant

FLCC outperforms AI Assistant on almost every programming language available

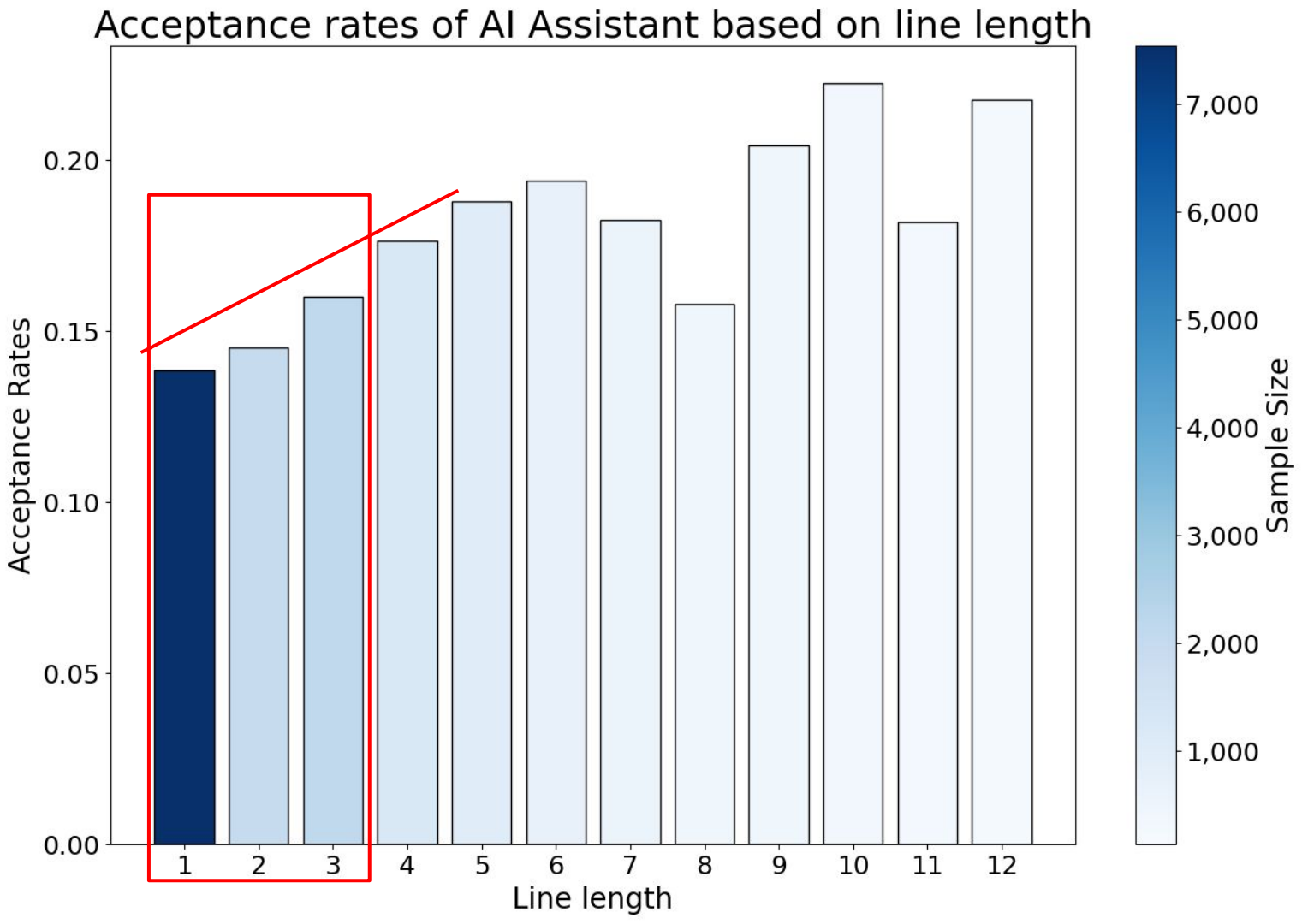


Why does FLCC outperform AI Assistant

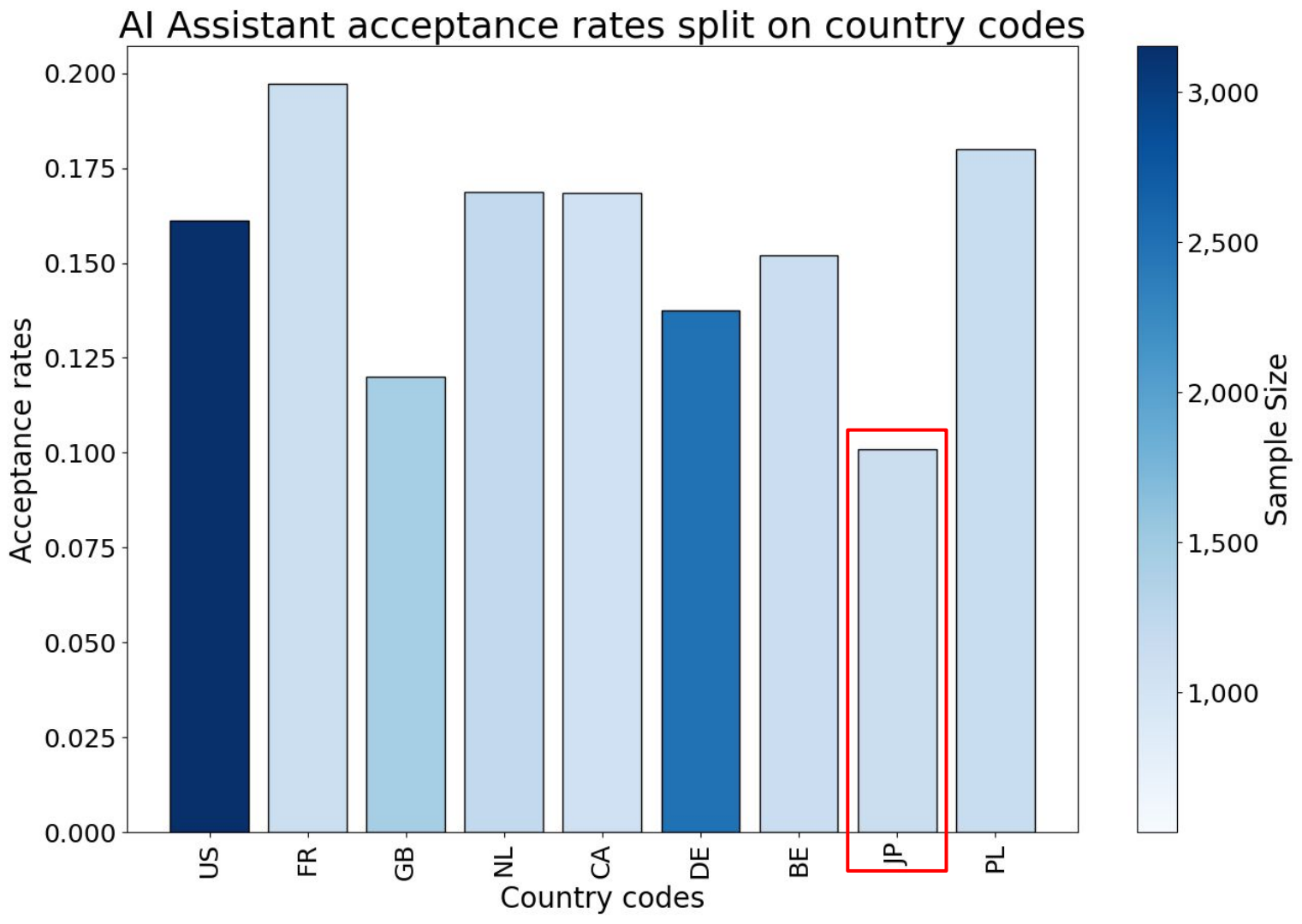
- Differences between FLCC and AI Assistant:
  - FLCC's context window is a lot smaller, and only includes everything in the current file above the caret
  - FLCC only suggests single line completions AI Assistant can suggest longer completions
- Suggesting longer completions should statistically lead to a lower acceptance rate
- Hypothesis:
  - AI Assistant performs less well due to the fact that completions are longer

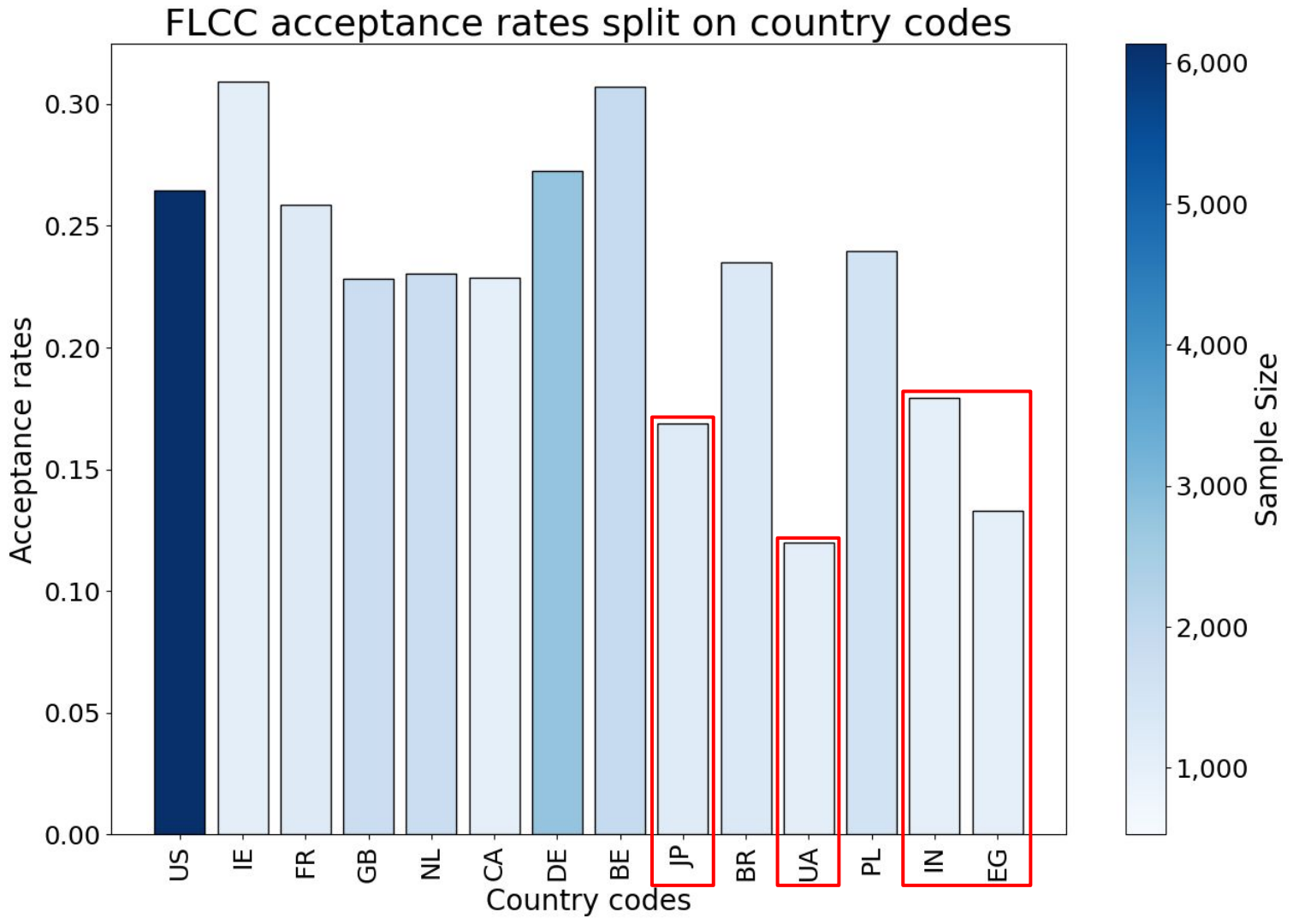
AI Assistant overwhelmingly suggests small and mostly single line completions

The single line completions also perform the worst



Non-Latin Script users have lower acceptance rates


AI Assistant acceptance rates split on country codes
Acceptance rates
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
US
FR
GB
NL
CA
DE
BE
JP
PL
Country codes
Sample Size
3,000
2,500
2,000
1,500
1,000


FLCC acceptance rates split on country codes
Acceptance rates
0.00
0.05
0.10
0.15
0.20
0.25
0.30
US
IE
FR
GB
NL
CA
DE
BE
JP
BR
UA
PL
IN
EG
Country codes
Sample Size
1,000
2,000
3,000
4,000
5,000
6,000

Why do Non-Latin Script country code users see decreased performance?

- Existing literature for general LLMs seems to indicate poorer performance for non-English source
- The country codes can indicate that comments and naming are done in the language of that country
- FLCC has higher absolute performance, a small context window giving less opportunity for the LLM to be ‘confused’ by the different language

Concluding remarks

Conclusion

- The user behavior change we see can be interpreted positively or negatively, future research is needed to be conclusive, although literature hints towards improvement for users (RQ1)
- Code executions dropping most likely hints to a reduced amount of testing when combined with existing works on testing when LLM support is available (RQ2)
- Benchmarks for comparing programming languages are limited, but the metrics do not seem to reflect real world (RQ3)

Future works

- Investigate why session length increase but code executions per hour drop
- Do the number of tests written actually decrease with LLM assistance
- Why AI Assistant seems to prefer single line or shorter completions
- Controlled study of what the impact of Locale is on code generations

Questions?

TU Delft

Q&A

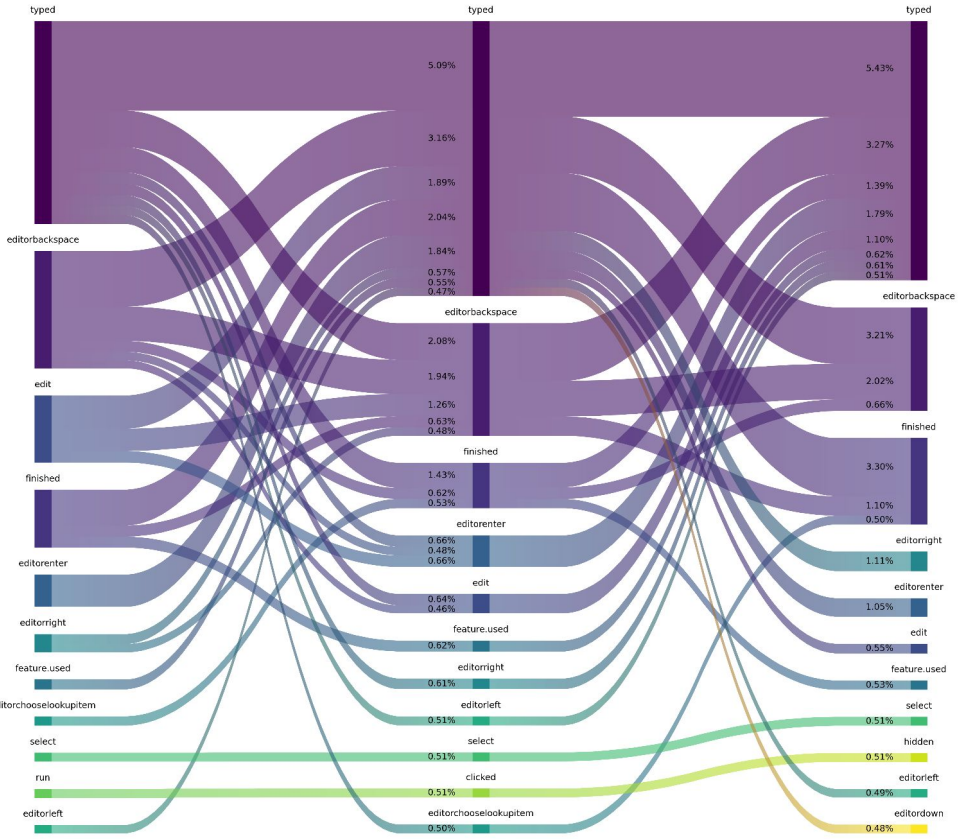
TU Delft

Threats

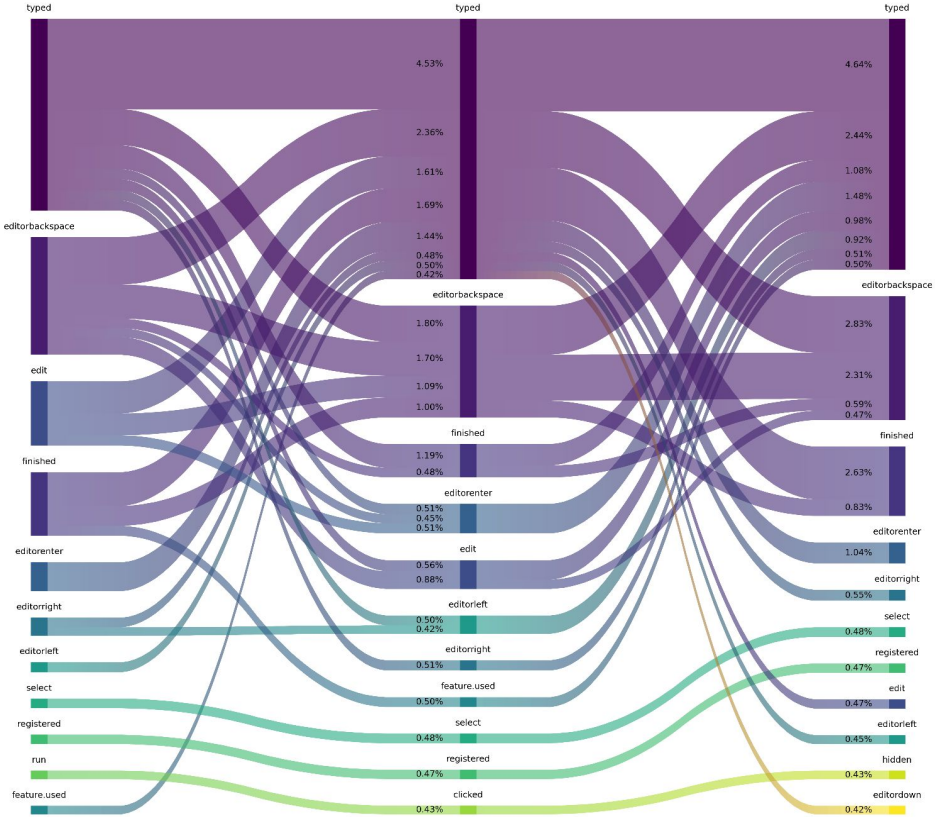
- Differences between IDEs can have skewed some results
- External LLM tools like ChatGPT are not accounted for in this study
- Missing or misattributed actions for user groups
- Acceptance rates do not take immediate deletion or undo’s into account
- Selected benchmark results for GPT-3.5-Turbo do not translate to AI Assistant performance

Microanalysis

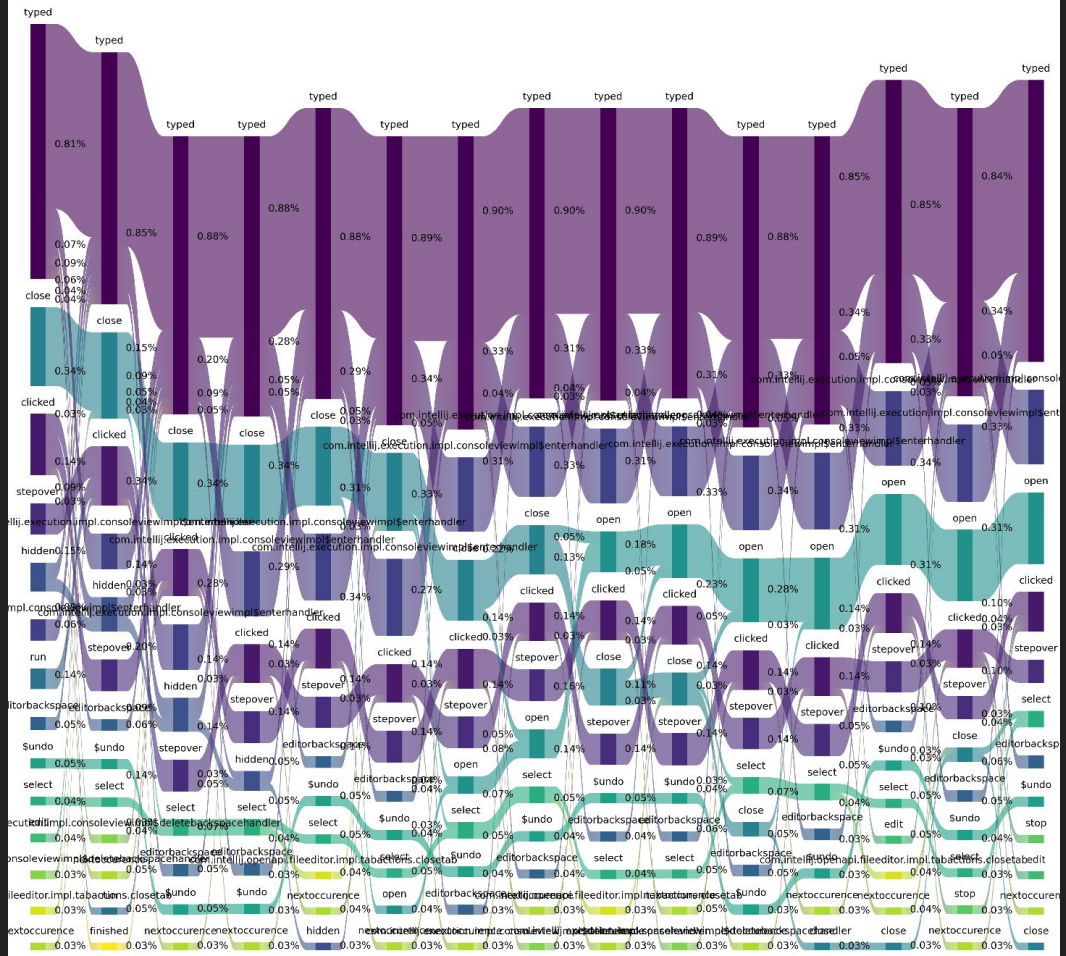
CL - Top 50 3-grams for Non-LLM or FLCC Users



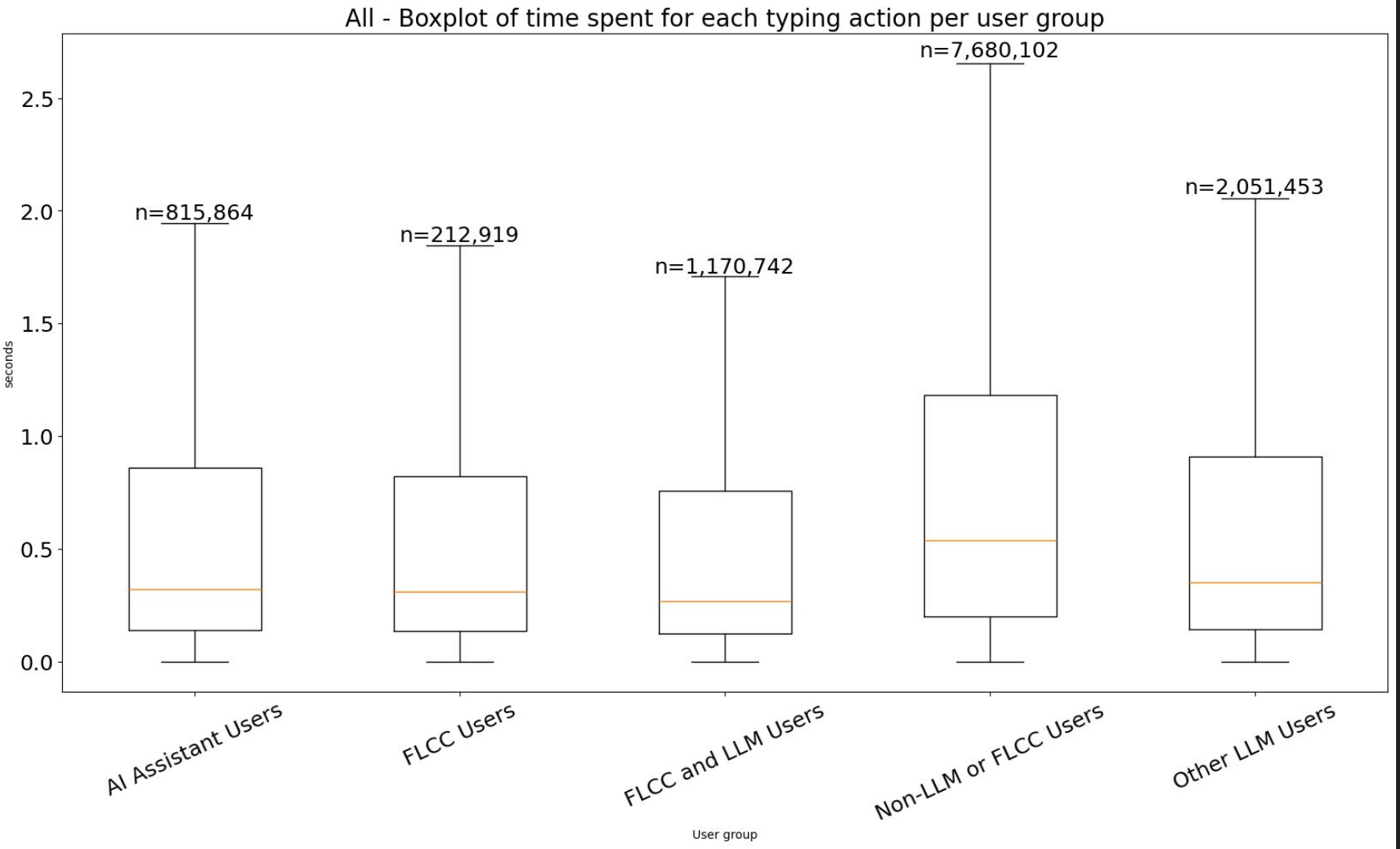
CL - Top 50 3-grams for AI Assistant Users

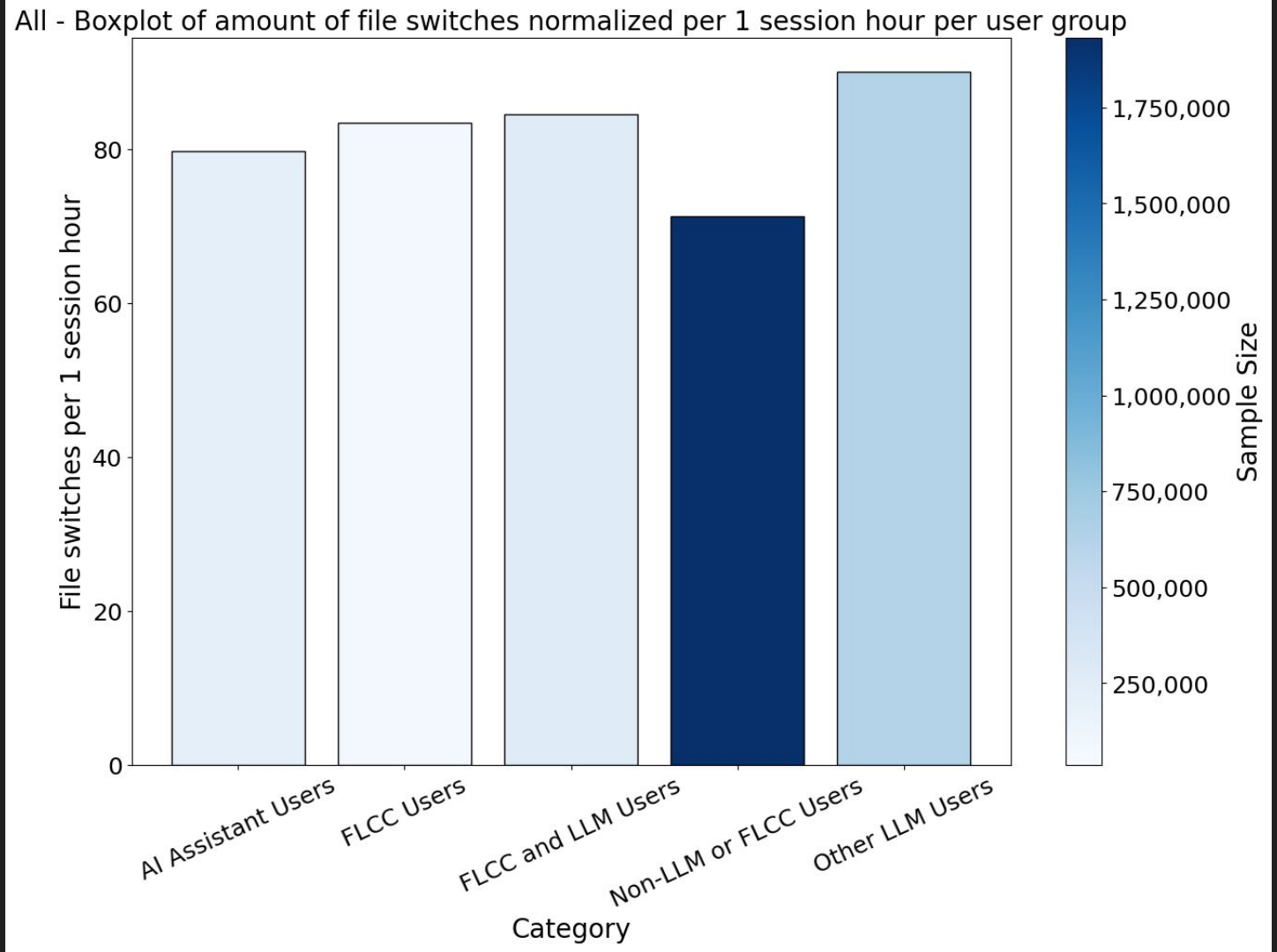


CL - Top 50 15-grams for Non-LLM or FLCC Users

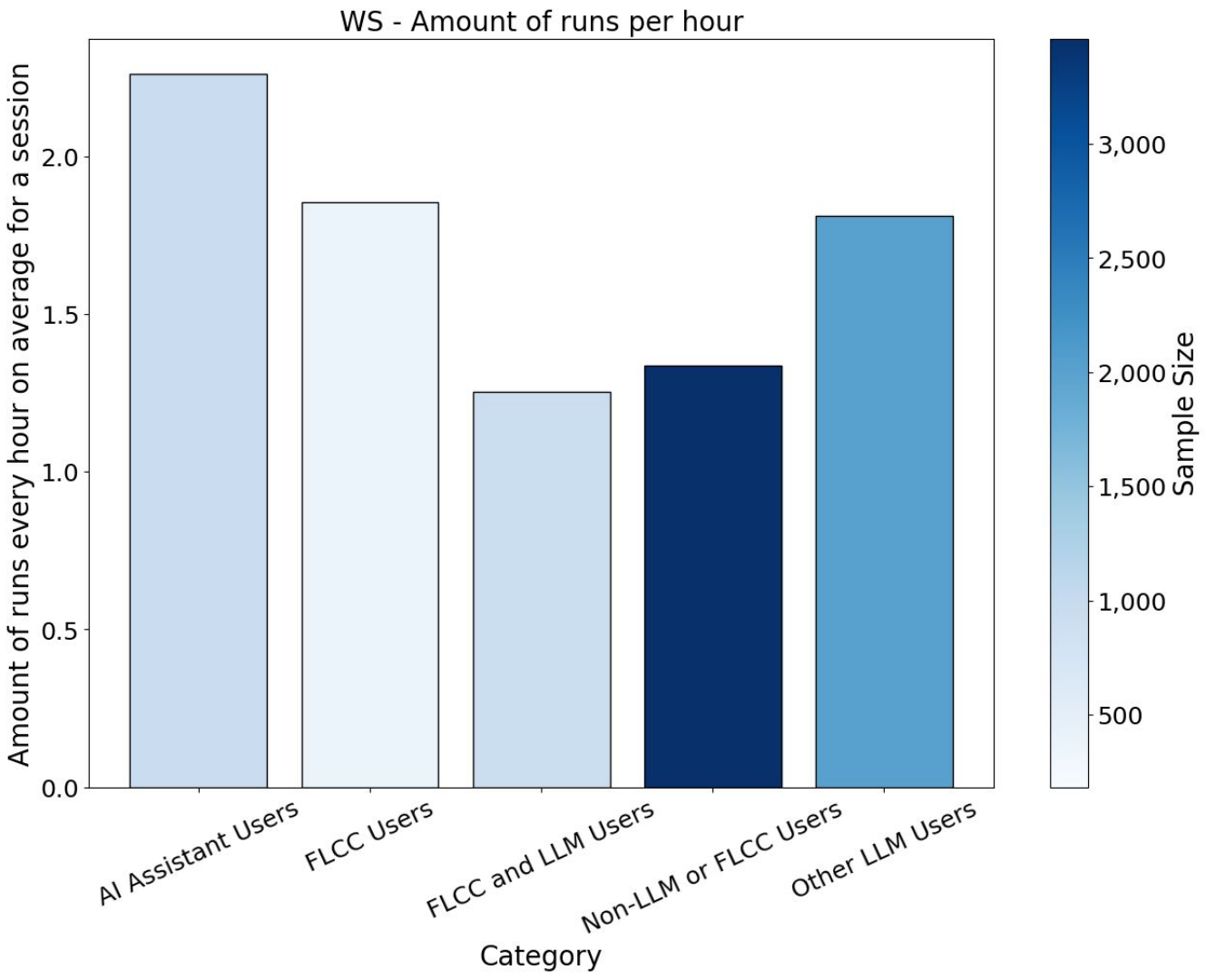


Macroanalysis


All - Boxplot of time spent for each typing action per user group
n=815,864
n=212,919
n=1,170,742
n=7,680,102
n=2,051,453
2.5
2.0
1.5
1.0
0.5
0.0
seconds
AI Assistant Users
FLCC Users
FLCC and LLM Users
Non-LLM or FLCC Users
Other LLM Users
User group


All - Boxplot of amount of file switches normalized per 1 session hour per user group
File switches per 1 session hour
0
20
40
60
80
AI Assistant Users
FLCC Users
FLCC and LLM Users
Non-LLM or FLCC Users
Other LLM Users
Category
Sample Size
1,750,000
1,500,000
1,250,000
1,000,000
750,000
500,000
250,000

Code executions


WS - Amount of runs per hour
Amount of runs every hour on average for a session
0.0
0.5
1.0
1.5
2.0
AI Assistant Users
FLCC Users
FLCC and LLM Users
Non-LLM or FLCC Users
Other LLM Users
Category
Sample Size
500
1,000
1,500
2,000
2,500
3,000