MigMate: A VS Code Extension for LLM-based Library Migration of Python Projects

Matthias Kebede
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates
msk9862@nyu.edu

May Mahmoud
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates
m.mahmoud@nyu.edu

Mohayeminul Islam
University of Alberta
Edmonton, Alberta, Canada
mohayemin@ualberta.ca

Sarah Nadi
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates
sarah.nadi@nyu.edu

Abstract

Modern software relies heavily on third-party software libraries to streamline the development process. The act of switching one library for a similar counterpart, called library migration, naturally occurs as libraries become outdated or unsuitable for the project. Manually migrating from one library to another is a time-consuming task. Our previous research developed MigrateLib, a command-line LLM-based migration tool that can automate the complete migration process. In this paper, we present our open-source VS Code IDE plugin, MigMate, that builds on MigrateLib by integrating the automated migration process into the developer's existing development environment. MigMate provides an interactive experience, allowing developers to view and confirm changes before they are applied. A preliminary user study shows that plugin usage consistently reduces the time taken to complete a library migration task, and it scores highly on the System Usability Scale.

CCS Concepts

• **Software and its engineering** → **Software maintenance tools**; **Integrated and visual development environments**; • **Human-centered computing** → Usability testing.

Keywords

VS Code, IDE plugin, LLM, library migration, third-party libraries

1 Introduction

Modern software relies on third-party libraries to streamline development. Libraries provide reusable code and abstraction, but also introduce new maintenance challenges [1]. Managing libraries can mean updating a version or performing a full *library migration*, where one library is replaced by another. Library migrations are difficult and time-consuming when done manually [13], as developers must learn the Application Programming Interfaces (APIs) of both libraries and then perform code transformations across their codebase. Although some tools recommend replacement libraries or map APIs between the two libraries [7, 9, 20, 21], they still leave significant manual effort for the developer.

Accordingly, both ourselves [10] and other researchers [2] have turned to investigating whether Large Language Models (LLMs) can be used to fully automate the library migration process, including both API mapping and code transformation. Our initial empirical study showed that LLMs can correctly perform a high portion of migrations in benchmark evaluations [10]. Based on the insights gained from the empirical study, we developed an LLM-based end-to-end Command Line Interface (CLI) for library migrations, MigrateLib, and evaluated it on additional repositories [11].

While standalone tools can be quite powerful, they add friction to the development process by forcing developers out of their workflow [15]. Ideally, such tools should be integrated into an Integrated Development Environment (IDE) to provide convenience and improve usability. To enable more streamlined usage of MigrateLib, in this paper, we develop a Visual Studio Code (VS Code) plugin called MigMate. Under the hood, MigMate uses the core functionality of MigrateLib, but adds an interactive UI that integrates into the developer workflow. It allows developers to view any suggested modifications to their code and approve or reject individual changes, giving the developer more control over the migration process and building trust in the tool. In doing so, our work contributes a human-in-the-loop approach to automated library migration that addresses key usability challenges.

We open-source MigMate at https://github.com/sanadlab/MigMate. We also provide a video demonstrating how MigMate works at https://www.youtube.com/watch?v=LHEmUFFz8_o.

2 Background and Related Work

Library migration is a process where a developer replaces a *source library* with a *target library* that provides similar functionality, without changing the behavior of the project. This requires updating the dependencies, updating the API usage in the code, and verifying behavior. Migrations typically happen between *analogous library pairs*, such as `requests` [19] and `httpx` [5], which overlap in functionality but differ in syntax and configuration [7, 9].

Researchers have developed tools to recommend or compare analogous libraries. MigrationAdvisor [9] provides Java recommendations via a web application, while LibComp [7] integrates suggestions into IntelliJ IDEA. These tools lower the burden of selecting a target library but focus mainly on recommendations, not code transformation. API mappings can also be discovered by

Conference'17, Washington, DC, USA
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

mining repositories [20] or analyzing documentation [21]. Other approaches like SOAR [18] use synthesis and error messages for automated migration, while recent work explores applications of LLMs for library migration [2, 10].

In our own previous work [10], we used Llama 3.1 (70B), GPT-4o mini, and GPT-4o on a subset of the code changes found in PyMigBench [12], a benchmark of real-world Python library migrations mined from open-source repositories. This empirical study compared LLM-generated migrations against developer implementations to assess correctness. A code change is considered correct if it exactly matches the developer's change or is manually identified as a valid alternative. A migration is considered partially correct if only some developer changes were matched, or fully correct if all changes were matched without making any additional refactoring changes. The results showed that GPT-4o achieved the highest accuracy, with 94% of migrations containing at least one correct code change and 57% of migrations being fully correct. GPT-4o mini performed similarly with 93% of migrations partially correct and 49% fully correct, while Llama 3.1 trailed behind with 51% of migrations partially correct and 26% fully correct. This evaluation shows that LLMs are already capable of handling complex migrations, with room for improvement.

Using the insights from our empirical study, we designed a CLI tool, MigrateLib, that combines LLMs with pre- and post-processing steps (see Sec. 3) to improve migration correctness [11]. We find that MigrateLib can migrate 32% of the migrations with complete correctness. Of the remaining migrations, only 14% of the migration-related changes are, on average, left for developers to fix. This need for human intervention motivated us to build an IDE plugin to allow for more seamless interaction.

Beyond technical methods of library migration, designing usable developer tools is crucial for integrating the automated process into the IDE. Research shows that developers need encouragement to trust in automated refactoring tools, with clear previews of changes before applying them [4]. Visualization studies further suggest that inline displays with optional panels are preferred [15]. We adopt these insights in our design of MigMate by combining automated support with human oversight to improve adoption and trust.

3 MigMate Design

Workflow and UI. We use guidelines from research on plugin usability [4, 15] and VS Code's documentation [16, 17] to design MigMate to be intuitive and efficient. We specifically design these features:

- **Context-Aware Activation:** MigMate loads lazily using VS Code Activation Events [16]. It activates when a workspace contains Python source files or a recognized dependency file (e.g., *requirements.txt*, *pyproject.toml*). This approach prevents unnecessary resource usage when the plugin is not explicitly needed.
- **Plugin Configuration:** MigMate provides flexible configuration options through the VS Code *Settings* interface, allowing developers to adjust plugin behavior to suit their workflows.
- **Seamless Migration Trigger:** Developers can start a migration directly from the dependency file with a hover or context menu. These triggers are easily accessible and keep the workflow entirely within the IDE, mitigating the burden of context-switching [15].
- **Guided Library Selection:** A Quick Pick [17] menu lists source libraries and subsequently prompts the developer to input the name of a target library. Quick Picks are a perfect choice for this step since they work well to facilitate short multi-step inputs.
- **Automated Migration Execution:** MigMate performs the migration, runs the project's test suite, and generates a final preview. It also shows a progress bar during the migration and warns the user in the event of migration errors or test failures.
- **Interactive Migration Preview:** Developers can review changes before applying them selectively. This step addresses known LLM issues such as unrelated edits [6, 10], increasing the developer's trust in and control over the tool.
- **Test Results View:** A Webview [17] displays a summary of migration test results. In the event that one or more tests fail after migration, a warning notification will prompt the user to open the Webview, helping developers investigate failures and identify the root cause.

CLI Overview. MigMate uses MigrateLib as the backend, so we first briefly discuss it and its use in our plugin.

MigrateLib [11] takes the names of the desired source and target library as its primary arguments, then proceeds through multiple iterative rounds. The initial *premig* round simply establishes a baseline for the project by running the existing test suite. The *llmmig* round sends the relevant code to an LLM and retrieves migrated content to be applied to the code. Subsequent rounds focus on refining the migration and ensuring its correctness.

MigrateLib leverages test results to determine the status of the migration. After the LLM migration, MigrateLib compares the test results before and after migration. If the results are the same, it considers the migration to be correct and does not proceed any further. Otherwise, MigrateLib runs two post-processing rounds trying to correct the migrations (specifically re-including code where the LLM says that the rest of the code stays the same and adding the async keyword to function definitions that use asynchronous libraries), and similarly runs the tests to verify the migration. MigrateLib preserves the test reports of each round for the user to review.

Integration with MigrateLib. MigMate acts as an intermediary between VS Code and the underlying MigrateLib CLI tool [11]. When starting a migration, MigMate spawns a child process to run MigrateLib in the background. The source and target libraries are passed to MigrateLib's initiating command, along with certain configuration options as additional arguments. MigMate then parses the migration output for use in the Migration Preview and Test Results views.

4 MigMate Workflow

In order to illustrate the expected developer workflow, we present a sample migration from `requests` to `httpx` using MigMate. The following are the high-level steps involved, which are explained in detail in the relevant subsections and shown in Fig. 1.

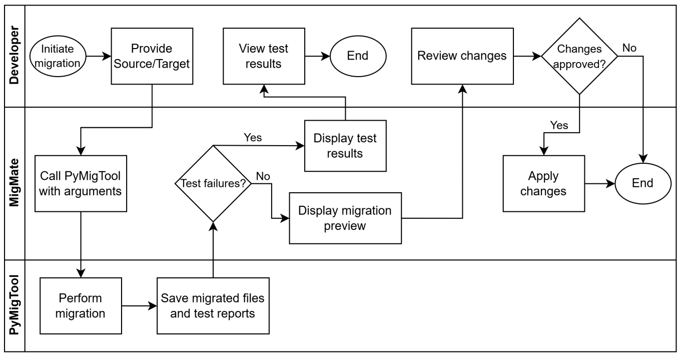


Figure 1: MigMate Workflow

(1) **Activate Plugin:** The developer opens a Python project in VS Code. MigMate automatically activates after detecting the appropriate source files.
(2) **Initiate a Migration:** The developer then opens the dependency file (*requirements.txt*) and hovers over *requests*, then selects `Migrate requests` (Fig. 2).
(3) **Select Libraries:** *requests* becomes the source library for the migration. The developer enters *httpx* into the provided text input to set the target library.
(4) **Run Migration:** MigMate calls MigrateLib using the selected libraries and current configuration. A progress bar is displayed in the notification area.
(5) **View Test Results:** MigMate parses MigrateLib's test report files and provides a summary comparing pre-migration test results to post-migration results (Fig. 5).
(6) **Review Proposed Changes:** MigMate opens a Migration Preview window (Fig. 4). The developer selects the desired changes to be applied, then closes the preview.
(7) **Apply Changes:** MigMate applies the approved changes to the source files.

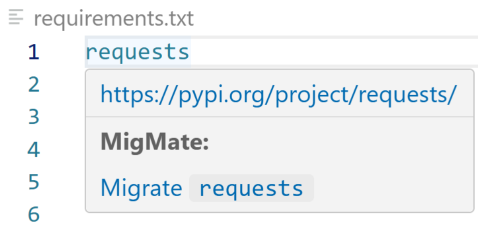
4.1 Initiating Migration

MigMate provides two main ways to initiate a migration. Given that the dependency file contains the Python libraries used in the project, it serves as a natural starting point for the migration process. The first approach involves right-clicking within the dependency file to bring up a context menu that provides a migration command. Alternatively, the user can move the cursor over the name of a library in the dependency file and wait for a hover menu to appear (Fig. 2).

Regardless of the trigger used, MigMate will present a Quick Pick menu (Fig. 3) to the user where they can input the source and target library names. When using the hover trigger, the source library will automatically be set as the library used to initiate it. Developers can also initiate a migration directly through the Command Palette. Selecting the 'MigMate: Migrate a Library' command will open a Quick Pick just as when using the context menu trigger.

4.2 Test Results

MigMate also offers a test result viewer, which the user is prompted to open in the event that any tests fail at the end of the migration



```
requirements.txt
requests
```

Figure 2: Hover Trigger

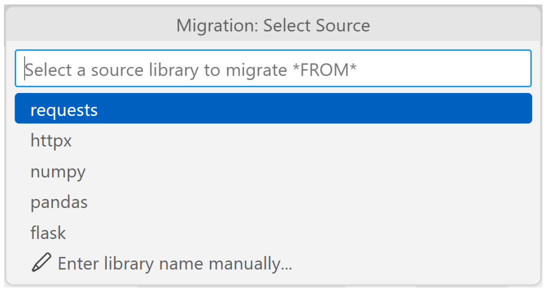

Migration: Select Source
Select a source library to migrate *FROM*
requests
httpx
numpy
pandas
flask
Enter library name manually...

Figure 3: Source Library Selection

process. This feature is implemented as a Webview, an embedded browser panel controlled by the plugin. The viewer reads the test report files produced by MigrateLib and renders the data as formatted HTML (Fig. 5). This allows developers to view a summary of pre- and post-migration test results, the specific error messages of each test, and the contents of the log file created during migration. Clicking the 'Go' button next to each test navigates to the relevant test file.

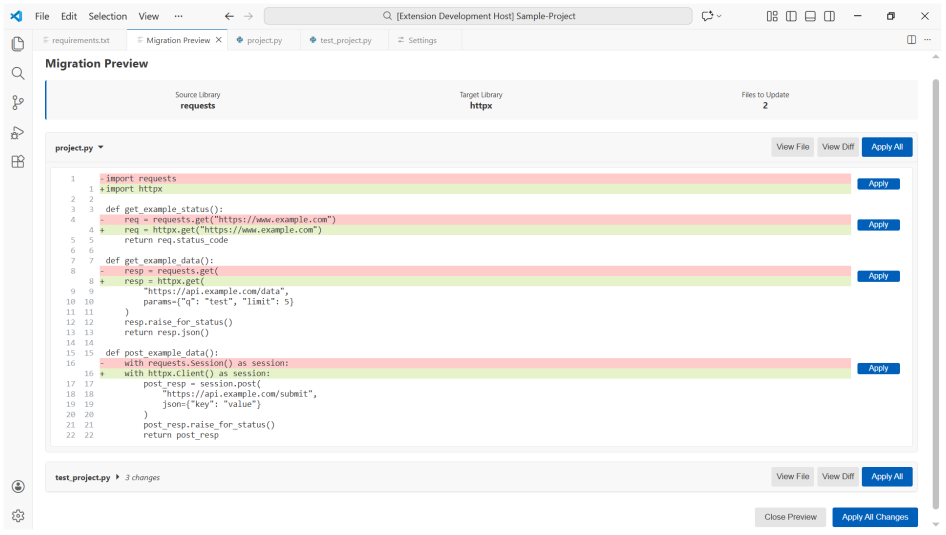
4.3 Migration Preview

A key feature of MigMate is that it requires explicit user approval before applying any migration changes. To generate the preview, MigMate compares the unmodified workspace files with the migrated copies saved by MigrateLib. Two preview styles are available, each offering a convenient interface for reviewing and managing changes across multiple files.

The first style leverages VS Code's built-in Refactor Preview window, which lists all suggested modifications by file. Developers can selectively enable or disable individual changes using checkboxes, providing fine-grained control over which are accepted. Upon confirmation, MigMate applies all approved modifications in a single bulk edit. Unselected changes are safely discarded without impacting the code.

Alternatively, a custom Webview interface (Fig. 4) displays a collapsible list of files and supports the incremental application of edits. With this style, developers can apply individual changes, all changes within a single file, or the entire migration at once. If the migration is accepted as a whole, the Webview will close automatically. Otherwise, the user manually exits by selecting the 'Close Preview' button once they are satisfied with the migration changes that they have already applied.

Both preview styles enable detailed selection of edits but differ in terms of when those edits are performed. The Refactor Preview



Migration Preview

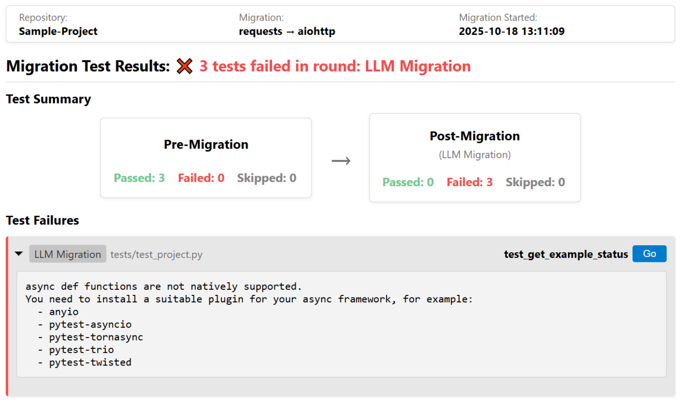
Source Library: **requests**
Target Library: **httpx**
Files to Update: **2**

project.py

```
-import requests
+import httpx

 def get_example_status():
-    req = requests.get("https://www.example.com")
+    req = httpx.get("https://www.example.com")
     return req.status_code

 def get_example_data():
-    resp = requests.get(
+    resp = httpx.get(
         "https://api.example.com/data",
         params={"q": "test", "limit": 5}
     )
     resp.raise_for_status()
     return resp.json()

 def post_example_data():
-    with requests.Session() as session:
+    with httpx.Client() as session:
         post_resp = session.post(
             "https://api.example.com/submit",
             json={"key": "value"}
         )
         post_resp.raise_for_status()
         return post_resp
```

test_project.py *3 changes*

Figure 4: Migration Preview (Webview) for *requests* to *httpx*



Repository: **Sample-Project**
Migration: **requests → aiohttp**
Migration Started: **2025-10-18 13:11:09**

Migration Test Results: ❌ 3 tests failed in round: LLM Migration

Test Summary

Pre-Migration
Passed: 3 Failed: 0 Skipped: 0

→

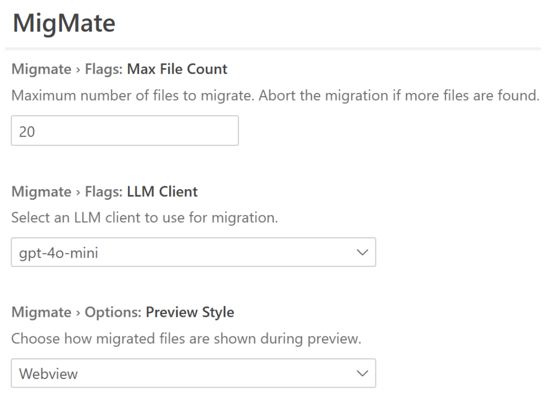
Post-Migration
(LLM Migration)
Passed: 0 Failed: 3 Skipped: 0

Test Failures

LLM Migration tests/test_project.py **test_get_example_status**

```
async def functions are not natively supported.
You need to install a suitable plugin for your async framework, for example:
  - anyio
  - pytest-asyncio
  - pytest-tornasync
  - pytest-trio
  - pytest-twisted
```

Figure 5: Migration Test Results



MigMate

Migmate › Flags: **Max File Count**
Maximum number of files to migrate. Abort the migration if more files are found.
20

Migmate › Flags: **LLM Client**
Select an LLM client to use for migration.
gpt-4o-mini

Migmate › Options: **Preview Style**
Choose how migrated files are shown during preview.
Webview

Figure 6: Plugin Configuration

aggregates approved changes into one confirmed update, whereas the Webview applies them incrementally as the developer proceeds. In either case, MigMate ensures that every modification is explicitly approved, preventing unintended modifications by the LLM and reinforcing developer trust in the migration process.

4.4 Configuration

Before performing a migration, developers may wish to customize how MigMate operates. Configuration options are accessible through the VS Code *Settings* interface (Fig. 6) or by editing a project-level *settings.json* file. The available parameters are organized into two categories based on their purpose:

Migration Flags: These settings mirror the command-line arguments of MigrateLib and control how migrations are executed. One notable item is the `--llm` flag, which allows the user to pass the name of the LLM model that they wish to use. MigrateLib currently supports OpenAI models, with GPT-4o mini being the default model, and checks for an `OPENAI_API_KEY` environment variable.

Extension Options: These configurations adjust plugin-specific usability features. Developers can choose between the Webview or Refactor Preview as their preferred migration preview method and toggle whether a preview is shown in the event of post-migration test failures.

5 Preliminary Evaluation

To evaluate the usability of MigMate, we conduct a preliminary small-scale user study with undergraduate students at NYU Abu Dhabi. We ask participants to perform library migration tasks both using MigMate and manually while collecting their feedback.

5.1 Study Design

We provide participants with a Python project that makes use of two libraries, *requests* and *tablib*, which can be replaced with *httpx* and *pandas*, respectively. We refer to these as Pair A (*requests* ⟶ *httpx*) and Pair B (*tablib* ⟶ *pandas*). After completing a short warm up

Table 1: Participant Grouping

Group	1st Task	Lib Pair	2nd Task	Lib Pair
A1	Manual	A	Assisted	B
A2	Assisted	A	Manual	B
B1	Manual	B	Assisted	A
B2	Assisted	B	Manual	A

task to familiarize them with the project's code, we ask participants to perform library migration tasks as follows:

- **Manual Migration:** Participants perform one migration task without using MigMate. They may access API documentation and other online resources, but cannot use LLMs to assist them. This serves as a baseline measurement.
- **Plugin-Assisted Migration:** Participants perform a second different migration task with the help of MigMate. They may additionally make use of the same resources as in the manual migration task.

5.1.1 Tasks. For Pair A, the project contains 14 *requests* usages spread across 2 files. Most of these are one-to-one changes that replace individual function calls and imports with their *httpx* counterparts. In addition, there are a few more complicated many-to-one changes due to inherent differences between the two libraries. Importantly, migrating this pair does not require any async usage. For Pair B, there are 22 *tablib* usages across 2 files, including a few one-to-one changes but primarily focusing on a mix of many-to-one and many-to-many changes.

For each task, participants have a maximum of 30 minutes to complete the migration. Before starting the plugin-assisted migration, we verbally instruct the participants on how to use the plugin and show them the README file for MigMate. The exact project and task instructions used are available at https://github.com/sanadlab/MigMate-Study-Repo.

5.1.2 Survey. After completing both tasks, we ask participants to fill in a short survey that contains Likert-type questions on perceived usability of MigMate based on the System Usability Scale (SUS) [3], a widely used questionnaire that measures perceived usability. The survey also includes optional feedback questions to gather qualitative data regarding the participants' experience with MigMate.

5.1.3 Participants Block Assignment. We conduct our study as a within-subjects design [8], where all participants are exposed to both migration tasks. We balance the participants across different setups (experimental blocks) of the study to mitigate learning effects, especially since the same project is used across both tasks. Half of the participants perform the manual migration first, while the other half begin with the plugin-assisted migration. We further divide those groups into those that start with Pair A and those that start with Pair B, for a total of four groups. Table 1 shows the four different experimental block configurations.

5.1.4 Collected data. We collect the time taken for each task, as well as telemetry data to identify usage patterns. Note that we do not measure the correctness of the migrations as our main focus is on understanding the participants' experience in using the plugin.

Table 2: Average Time to Complete Migration (min)

Lib Pair	Manual	Plugin
(A) requests → httpx	25:23	10:42
(B) tablib → pandas	27:51	10:48

5.2 Participant Recruitment

We recruited nine participants for the study, but one withdrew before completing the session. The remaining eight participants consisted of two first-year students, a third-year student, four fourth-year students, and one recent alumnus. All participants self-rated their Python skills and familiarity with VS Code on a 5-point Likert scale, and we included only those who indicated at least a 3 in both areas.

5.3 Results

Table 2 shows the average time taken to solve each task manually versus using MigMate. We find that across both tasks, the manual migration took more time to complete than the assisted one. We can see that the plugin saves 60% of the time required on average.

Based on the telemetry data, we find that the Hover Trigger was the most frequently used method for initiating a library migration by participants (62.5%), followed by the Context Menu (33.3%). This matches our expectations, as the hover trigger reduces friction by eliminating the step of selecting a source library for the current migration. The Command Palette was used only once, suggesting it may be less intuitive for users. We find that four participants used the migration-level changes, three used the file-level changes, while one participant completed the migration by exclusively making individual changes. This result does not indicate the exact reasoning behind these choices, which could be due to confidence or simply personal preference. We need further investigation to better understand if a particular granularity level has advantages/preferences.

Based on the questionnaire, we find that MigMate's mean SUS usability score is 80.9 on the 100-point scale, placing it around the 90th percentile and earning an A-grade [14]. Six of the participants also left comments and suggestions in the optional feedback. One recurring request was to implement library recommendations during the target library selection when initiating a migration. Another common point was an appreciation for the preview's clarity, with a few participants suggesting that it should also indicate the confidence in each suggested change.

5.4 Threats to Validity & Future Work

Our current evaluation is a small-scale preliminary evaluation of MigMate. Specifically, our sample size of 8 participants is rather small and the four experiment blocks do not all have the same number of participants. Specifically, Group B1 has three members while Group A2 has only one due to random assignment and some participants canceling their sessions. Another threat is that the participants had limited exposure to MigMate's features. Since they only used the plugin to complete one migration task, they did not need to interact with the configuration options, and most of the participants never experienced test failures during an assisted migration. While we currently have initial positive results about MigMate's usability, future larger scale evaluations should include migration tasks that are known to result in failing tests and to allow participants to experiment with the configuration options. Such an extended evaluation can also helps us determine useful features to add to MigMate, as well as how to improve the existing functionality.

6 Conclusion

This paper presented MigMate, a VS Code extension that integrates MigrateLib [11], an LLM-based migration tool, to support semi-automated Python library migration. The system combines automated code transformation with interactive review of LLM-generated code changes, allowing developers to selectively apply suggested changes directly in their workspace. By embedding this process into the IDE, MigMate streamlines migration while maintaining developer control over code modifications. Our preliminary evaluation suggests that developers appreciated having a preview of the exact changes that will happen. More broadly, this work demonstrates how automation can be applied in ways that preserve transparency and confidence in AI-assisted tools.

Acknowledgments

We would like to thank Ajay Kumar Jha for providing feedback in the early stages of this study.

References

[1] Rabe Abdalkareem. 2017. Reasons and drawbacks of using trivial npm packages: the developers' perspective. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 1062–1064. doi:10.1145/3106237.3121278

[2] Aylton Almeida, Laerte Xavier, and Marco Tulio Valente. 2024. Automatic Library Migration Using Large Language Models: First Results. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Barcelona, Spain) *(ESEM '24)*. Association for Computing Machinery, New York, NY, USA, 427–433. doi:10.1145/3674805.3690746

[3] John Brooke et al. 1996. SUS - A quick and dirty usability scale. In *Usability Evaluation in Industry*. London: Taylor and Francis, 189–194.

[4] Dustin Campbell and Mark Miller. 2008. Designing refactoring tools for developers. In *Proceedings of the 2nd Workshop on Refactoring Tools* (Nashville, Tennessee) *(WRT '08)*. Association for Computing Machinery, New York, NY, USA, Article 9, 2 pages. doi:10.1145/1636642.1636651

[5] Tom Christie and contributors. 2024. *HTTPX: A next-generation HTTP client for Python*. https://www.python-httpx.org/

[6] Jonathan Cordeiro, Shayan Noei, and Ying Zou. 2025. LLM-Driven Code Refactoring: Opportunities and Limitations. In *Proceedings of the 47th International Conference on Software Engineering* (Ottawa, Ontario, Canada) *(ICSE '25)*. IEEE Press.

[7] Rehab El-Hajj and Sarah Nadi. 2020. LibComp: an IntelliJ plugin for comparing Java libraries. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1591–1595. doi:10.1145/3368089.3417922

[8] Anthony G. Greenwald. 1976. Within-subjects designs: To use or not to use? *Psychological Bulletin* 83, 2 (1976), 314–320. https://doi.org/10.1037/0033-2909.83.2.314

[9] Hao He, Yulin Xu, Xiao Cheng, Guangtai Liang, and Minghui Zhou. 2021. MigrationAdvisor: recommending library migrations from large-scale open-source data. In *Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings* (Virtual Event, Spain) *(ICSE '21)*. IEEE Press, 9–12. doi:10.1109/ICSE-Companion52605.2021.00023

[10] Mohayeminul Islam, Ajay Kumar Jha, May Mahmoud, Ildar Akhmetov, and Sarah Nadi. 2025. An Empirical Study of Python Library Migration Using Large Language Models. In *Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE 2025)* (Seoul, Republic of Korea) *(ASE '25)*. Association for Computing Machinery, New York, NY, USA.

[11] Mohayeminul Islam, Ajay Kumar Jha, May Mahmoud, and Sarah Nadi. 2025. PyMigTool: a tool for end-to-end Python library migration. arXiv:2510.08810 [cs.SE] https://arxiv.org/abs/2510.08810

[12] Mohayeminul Islam, Ajay Kumar Jha, Sarah Nadi, and Ildar Akhmetov. 2023. PyMigBench: A Benchmark for Python Library Migration. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. 511–515. doi:10.1109/MSR59073.2023.00075

[13] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2018. Do developers update their library dependencies? *Empirical Softw. Engg.* 23, 1 (Feb. 2018), 384–417. doi:10.1007/s10664-017-9521-5

[14] James R. Lewis and Jeff Sauro. 2018. Item benchmarks for the system usability scale. *J. Usability Studies* 13, 3 (May 2018), 158–167.

[15] Xinhong Liu and Reid Holmes. 2020. Exploring Developer Preferences for Visualizing External Information Within Source Code Editors. In *2020 Working Conference on Software Visualization (VISSOFT)*. 27–37. doi:10.1109/VISSOFT51673.2020.00008

[16] Microsoft. 2025. *Visual Studio Code - Activation Events*. https://code.visualstudio.com/api/references/activation-events

[17] Microsoft. 2025. *Visual Studio Code - UX Guidelines*. https://code.visualstudio.com/api/ux-guidelines/overview

[18] Ansong Ni, Daniel Ramos, Aidan Z.H. Yang, Inês Lynce, Vasco Manquinho, Ruben Martins, and Claire Le Goues. 2021. SOAR: A Synthesis Approach for Data Science API Refactoring. In *Proceedings of the 43rd International Conference on Software Engineering* (Madrid, Spain) *(ICSE '21)*. IEEE Press, 112–124. doi:10.1109/ICSE43902.2021.00023

[19] Kenneth Reitz and contributors. 2025. *Requests: HTTP for Humans*. https://requests.readthedocs.io

[20] Cédric Teyton, Jean-Rémy Falleri, and Xavier Blanc. 2013. Automatic discovery of function mappings between similar libraries. In *2013 20th Working Conference on Reverse Engineering (WCRE)*. 192–201. doi:10.1109/WCRE.2013.6671294

[21] Zejun Zhang, Minxue Pan, Tian Zhang, Xinyu Zhou, and Xuandong Li. 2020. Deep-Diving into Documentation to Develop Improved Java-to-Swift API Mapping. In *Proceedings of the 28th International Conference on Program Comprehension* (Seoul, Republic of Korea) *(ICPC '20)*. Association for Computing Machinery, New York, NY, USA, 106–116. doi:10.1145/3387904.3389282