PySpark 和 PyFlink 中 Py4j 的作用：Python 与 Java 交互的桥梁

引言

Apache Spark 和 Apache Flink 是当前大数据处理领域中两个非常流行的分布式计算框架。它们都以其处理大规模数据集的能力而闻名，并广泛应用于批处理、流处理和机器学习等多种场景 [1]。这两个框架的核心都是使用 Java 或 Scala 这类运行在 Java 虚拟机（JVM）上的语言构建的。然而，为了更广泛地被数据科学家和工程师所采用，它们都提供了 Python 接口，即 PySpark 和 PyFlink [1]。Python 在数据科学领域拥有庞大的用户群体和丰富的生态系统，因此提供 Python API 对于提升 Spark 和 Flink 的易用性和集成性至关重要。一个核心问题随之而来：PySpark 和 PyFlink 如何实现其 Python API 与底层 Java/Scala 引擎的交互？答案是它们都依赖于一个名为 Py4j 的 Python 库。本文旨在深入探讨 PySpark 和 PyFlink 使用 Py4j 的原因，分析 Py4j 在它们架构中的作用，并探讨其在分布式计算中的更广泛应用。

Py4j 的原理：连接 Python 与 Java 的桥梁

Py4j 是一个 Python 库，它使得运行在 Python 解释器中的 Python 程序能够动态地访问 Java 虚拟机（JVM）中的 Java 对象 [12]。这意味着 Python 程序可以创建 Java 对象，调用 Java 方法，访问 Java 字段，并且能够像操作原生 Python 对象一样操作 Java 集合 [12]。例如，Python 程序可以实例化一个 java.util.Random 对象并调用其 nextInt() 方法来生成随机数 [13]。此外，Py4j 还支持回调机制，允许 Java 程序反过来调用 Python 对象的方法 [13]。这通过允许 Python 类实现 Java 接口来实现 [18]。Py4j 的核心运作方式是在 Python 解释器和 JVM 之间建立一个连接 [19]，这个连接通常通过本地网络套接字（socket）实现 [13]。

Py4j 实现跨语言互操作性的机制基于客户端-服务器模型。在 Java 进程中，需要启动一个 GatewayServer 实例，该实例监听来自 Python 程序的连接 [19]。Java 应用程序通常需要创建一个 GatewayServer 实例，并指定一个 Python 程序可以访问的入口点对象 [13]。在 Python 进程中，需要创建一个 JavaGateway 对象，该对象充当连接到 GatewayServer 的客户端，并提供访问 Java 对象的入口 [19]。Python 程序通过创建 JavaGateway 实例来建立与 JVM 的连接 [13]。默认情况下，GatewayServer 监听在 localhost:25333 这个地址和端口上 [19]。Python 和 Java 之间的通信通过一个基于文本的协议进行，该协议用于发送命令和数据。虽然这种协议对于常规的方法调用来说是相对高效的，但对于传输非常大的数值数据时可能会产生一定的开销 [19]。在过去，曾考虑过使用二进制协议（如 Protocol Buffers），但内部基准测试表明，对于 Py4j 经常交换的包含方法名、类名和变量名等大量小字符串的场景，文本协议在大小和时间效率方面表现更好 [19]。Py4j 的这种基于客户端-服务器模型和套接字通信的架构，使得 Python 和 Java 进程可以独立运行，甚至可以部署在分布式环境中的不同机器上。此外，这种分离的设计意味着 Python 代码不需要在 JVM 中执行，从而允许开发人员使用任何标准的 Python 库 [21]。

PySpark 的架构与 Py4j 的作用

PySpark 是 Apache Spark 的 Python API。Spark 的核心是用 Scala 和 Java 编写的，并在 JVM 上运行 [2]。PySpark 的出现使得开发人员可以使用 Python 来运行 Spark 作业，这得益于 Py4j 的强大功能 [2]。PySpark 的高层架构主要包含以下几个关键组件：

- 驱动程序（**Driver Program**）：运行用户编写的 Python 应用程序，负责创建 SparkContext 或 SparkSession，并协调 Spark 作业的执行 [14]。
- **SparkContext** 或 **SparkSession**：是 Spark 功能的入口点，负责初始化 JVM 进程 [3]。在驱动程序中，SparkContext 使用 spark-submit 命令在一个子进程中启动本地 Spark JVM 进程 [19]。
- 工作节点（**Worker Nodes**）：在弹性分布式数据集（RDDs）或 DataFrames 上并行执行任务 [1]。当需要将 Python 原生函数映射到其 Java 对应部分时，Python 工作进程会被延迟启动 [14]。
- **Spark** 引擎：底层使用 Scala/Java 编写，在驱动程序和工作节点上的 JVM 中运行 [2]。

当一个 PySpark 应用程序启动时，SparkContext 会在驱动程序所在的机器上初始化一个 JVM 进程，这个过程是通过执行 spark-submit 命令来完成的 [19]。随后，Py4j 被用于在 Python 驱动程序和这个本地 Spark JVM 之间建立一个通信网关 [2]。驱动程序通过读取一个临时文件中的连接信息（包括端口号和密钥）来建立 Py4j 网关 [20]。在 Python 中对 PySpark DataFrames 或 RDDs 执行的操作，会通过 Py4j 转换为对底层 JVM 中 Java/Scala 对象的相应操作 [29]。例如，PySpark DataFrame 的 _jdf 属性实际上是一个 JavaObject，它引用了 JVM 端的 DataFrame 实例 [34]。在转换操作中使用的 Python 函数（例如在 map 或 filter 中），会被序列化（通常使用 Pickle 或 cloudpickle），然后发送到 Spark 工作节点，在这些节点上，函数会在独立的 Python 进程中执行 [14]。Py4j 在 PySpark 中主要用于驱动程序端的控制平面通信，负责任务的编排和管理。而 Python 工作进程和 JVM 之间的数据传输，则可能通过其他机制（如 TCP 套接字）进行管理，这通常由 JVM 中的 PythonRDD 对象负责 [20]。

Py4j 在 PySpark 中扮演着至关重要的角色。Python 中的 SparkContext 对象通过 Py4j 持有一个对 JVM 中 JavaSparkContext 对象的引用 [35]。这使得 Python 能够访问 Spark 的 Java/Scala 入口点 [35]。在 Python API 中执行的操作（如创建 DataFrame 或调用转换操作），会通过 Py4j 网关转化为对相应 Java 对象的方法调用 [34]。例如，调用一个 Python DataFrame 的方法会自动触发 JVM 端 Scala DataFrame 的相应方法 [34]。Py4j 负责在这些调用过程中，根据需要将 Python 数据类型转换为其 Java 等效类型，反之亦然 [18]。简单的 Python 对象，如字符串或整数，可以很容易地通过 Py4j 在 Python 和 Java 之间传递 [14]。此外，JVM 中发生的错误会作为 Py4JJavaError 异常传播回 Python 驱动程序，为调试提供了有价值的信息 [18]。这些错误通常指示与 Java 相关的组件（如 Apache Iceberg）交互时出现的模式不匹配或序列化问题 [36]。总而言之，Py4j 是 PySpark 的核心组成部分，它使得高层 Python API 能够无缝地与底层使用 Java/Scala 编写的、经过性能优化的 Spark 引擎

进行交互。这种抽象使得数据科学家能够利用 Python 的易用性，同时受益于 Spark 的分布式处理能力。Py4j 在驱动程序端与 JVM 实例的通信是 PySpark 运行的基础 [14]。

PyFlink 的架构与 Py4j 的作用

PyFlink 是 Apache Flink 的 Python API。与 Spark 类似，Flink 的核心也是使用 Java 编写的，并在 JVM 上运行 [5]。PyFlink 的架构主要包括以下关键组件：

- 作业管理器（**JobManager**）：负责协调作业的执行和管理任务的分发 [8]。作业管理器接收作业图（JobGraph）并将任务分发给任务管理器 [8]。
- 任务管理器（**TaskManager**）：并行执行作业管理器分配的任务 [8]。任务管理器在 JVM 进程中运行 [8]。
- **Flink** 引擎：底层使用 Java 编写，在作业管理器和任务管理器上的 JVM 中运行 [5]。
- **PyFlink API**：提供 DataStream API 和 Table API，用于使用 Python 构建批处理和流处理应用程序 [5]。

与 PySpark 类似，PyFlink 也使用 Py4j 来实现 Python API 与 Flink Java 核心的通信 [6]。PyFlink 在其操作中高度依赖 Py4j [37]。当提交一个 PyFlink 作业时，Python 程序会使用 Py4j 与 JVM 中的作业管理器进行交互 [6]。在启动时，作业图会通过 Py4j 从 Python 作业中检索出来 [6]。Python 代码，包括用户自定义函数（UDFs），需要在 Flink 集群中执行。Py4j 在作业提交和协调的初始阶段起着关键的通信作用 [7]。当通过 flink run 提交 Python 作业以及编译包含 Python UDFs 的 Java/Scala 作业时，PyFlink 会使用 Python 解释器 [37]。对于 Python UDFs 的执行，Flink 利用 Apache Beam 的可移植性框架来管理 JVM 和 Python 虚拟机之间的执行环境和通信，这通常使用 gRPC 服务 [7]。Py4j 在建立这种交互中也发挥着作用 [6]。在 PyFlink 架构的本地部分，Python API 通过 Py4j 同步调用相应的 Java API [7]。

Py4j 在 PyFlink 中扮演着基础通信层的角色，尤其在作业编译和提交的初始阶段至关重要。PyFlink 中的 Python Table API 是 Java Table API 的映射，许多 Python 方法调用通常通过 Py4j 同步调用相应的 Java 方法 [7]。编写 Python API 的本质就是如何使用 Py4j 调用 Java API [7]。Py4j 用于从 Python 作业中检索作业图，并将其转换为在 Flink 集群上运行的等效作业 [6]。作业图包含了 Flink 作业的转换和配置信息 [8]。与 PySpark 类似，Py4j 在 API 交互过程中处理 Python 和 Java 之间的数据类型转换 [8]。Flink JVM 中发生的与 Python 代码交互相关的错误会作为 Py4j 异常报告回 Python 环境 [37]。一个常见的错误是 Py4JJavaError，它指示在调用 Java 方法时发生了问题 [37]。总而言之，在 PyFlink 中，Py4j 是实现 Python API 与核心 Flink 运行时交互的关键。PyFlink 中的 Python API 本质上是 Flink Java API 的一个包装器，而 Py4j 则促进了这种包装 [6]。

交互需求比较分析

PySpark 和 PyFlink 都需要一种机制，使得 Python API 能够控制和管理一个使用 Java/Scala 实现的分布式计算引擎 [1]。它们都需要将命令和配置从 Python 环境传递到

JVM [6]。两者都需要处理在分布式环境中执行用户提供的 Python 代码（UDFs），这需要 Python 和 Java 进程之间进行某种形式的交互 [7]。最后，两者都受益于一种将 Java 端发生的错误和异常传播回 Python 用户的方式 [18]。

尽管存在这些相似性，PySpark 与 Python 工作进程的交互（用于 RDD 转换）可能具有与 PyFlink 的 UDF 执行模型略有不同的数据序列化和通信模式，后者利用了 Apache Beam [7]。PySpark 工作进程通过 TCP 套接字将结果返回给 JVM [20]，而 PyFlink 的 UDF 执行涉及使用 gRPC 服务在 JVM 和 Python VM 之间进行数据和状态管理 [38]。此外，Flink 的原生流处理能力可能引入与 Spark 的微批处理方法不同的实时交互需求，但对 Python-Java 通信的核心需求依然存在 [4]。Flink 以低延迟实时处理每个事件，而 Spark 使用微批处理 [4]。

Py4j 的特性使其非常适合满足这两种框架的需求。它提供了一种直接的方式，让 Python 调用 Java 对象的方法并访问其属性，这对于控制底层的计算引擎至关重要 [12]。Py4j 对基本数据类型转换的支持简化了两种语言之间信息的交换 [18]。其基于客户端-服务器模型的套接字通信架构，为 Python 和 Java 进程提供了必要的独立性，这非常适合分布式环境 [13]。此外，Py4j 处理 Python 中 Java 异常的能力使得这两个框架都能提供更好的错误报告和调试体验 [18]。总而言之，PySpark 和 PyFlink 的根本需求是弥合用户友好的 Python API 和高性能 Java/Scala 后端之间的差距。Py4j 为这种核心需求提供了一个通用且相对简单的解决方案，这使得它成为这两个项目的自然选择。尽管它们的内部执行模型存在一些差异，但这两个框架的首要需求都是允许 Python 代码与基于 JVM 的系统进行交互和控制。Py4j 的核心功能直接满足了这一需求，它允许从 Python 动态访问 Java 对象。

Py4j 的替代方案

除了 Py4j 之外，还有其他一些 Python-Java 互操作库可供选择，包括：

- **Jython**：一个用 Java 实现的 Python，允许 Python 代码直接在 JVM 上运行 [21]。
- **JPype**：一个在 Python 进程中嵌入 JVM 的库，提供对 Java 类和对象的直接访问 [28]。
- **PyJnius**：一个用于访问 Java 类的 Python 库，主要关注 Android 开发 [41]。
- **Jep (Java Embedded Python)**：允许在 JVM 中嵌入 CPython 解释器 [41]。

下表总结了这些替代方案与 Py4j 的比较：

库名称	主要机制	主要优点	主要缺点	活跃开发状态	分布式计算适用性
Py4j	套接字	便携性好，支持回调，可控制多个 JVM，活跃开发，文档完善，被 PySpark 广泛	相比 Jython 和 JPype，可能存在更高的开销	活跃	高

		使用			
Jython	JVM 中的 Python	与 Java 集成紧密，JVM 内数据共享开销可能较低	与部分 CPython 库不兼容，可能落后于最新的 Python 特性，在 JVM 中处理 Python 对象可能存在性能问题	状态不确定	中等
JPype	Python 中的 JVM	计算性能好，支持在 Python 中实现 Java 接口，允许直接内存访问和使用 Java 线程	学习曲线可能较陡峭，可能存在 JVM 生命周期管理问题，在大数据框架领域可能不如 Py4j 活跃开发	活跃	中等
PyJnius	JNI	使用 Cython 编写，速度快，专注于从 Python 访问 Java	主要面向 Android 开发，处理原始数组和多维数据可能存在限制	活跃	低
Jep	JVM 中的 CPython	允许 Java 访问 Python 作为子解释器，支持 CPython 扩展	主要关注 Java 调用 Python，文档可能有限，部分 Python 模块在子解释器中可能无法正常工作	活跃	低

PySpark 和 PyFlink 倾向于选择 Py4j 的原因有以下几点：

- 成熟度和稳定性：Py4j 是一个成熟且稳定的库，在像 Apache Spark 这样的大规模项目中有着悠久的使用历史 [54]。
- 活跃开发：Py4j 一直在积极维护和更新，确保与新版本的 Python 和 Java 的兼容性 [48]。
- 回调支持：Java 回调 Python 对象的能力对于分布式框架中的某些功能至关重要 [13]。

- 进程独立性：Py4j 的设计允许 Python 和 Java 进程独立运行，这与 Spark 和 Flink 的分布式特性非常契合 [13]。这也允许使用任何 Python 库，包括那些带有 C 扩展的库 [21]。
- 便携性：使用标准套接字使得 Py4j 具有很高的便携性，可以跨不同的操作系统和网络配置运行 [28]。
- 与 **Spark** 的集成：PySpark 与 Py4j 的深度集成（Py4j 是 PySpark 的核心）很可能影响了 Flink 为 PyFlink 采用 Py4j 的决定，以保持一致性并利用大数据生态系统中现有的知识 [1]。PySpark 对 Py4j 的依赖使其成为一个经过充分测试和支持的解决方案 [54]。

尽管存在其他替代方案，但 Py4j 在成熟度、活跃开发、回调支持等关键特性方面的综合优势，以及其在 Apache Spark 中的成功应用，使其成为 PySpark 和 PyFlink 实现 Python-Java 交互最合适和最可靠的选择。Py4j 基于套接字的通信所提供的便携性也是在各种分布式环境中一个重要的优势 [28]。

Py4j 在分布式计算中的更广泛应用

Py4j 不仅限于 PySpark 和 PyFlink，它还是一个通用的库，可以用于任何需要 Python 与 Java 代码交互的 Python 应用程序，包括 Spark 和 Flink 之外的分布式系统 [13]。例如，Python 可以用于编排和控制基于 Java 的分布式服务，或者将基于 Python 的分析与基于 Java 的数据处理管道集成 [26]。例如，Py4j 被用于像 NXCALS 这样的项目中，以便在控制系统的上下文中从 Python 访问 Java API [49]。

Py4j 的多功能性使其成为构建和集成涉及 Python 和 Java 的分布式系统的宝贵工具，允许开发人员利用每种语言的优势。它使得在分布式环境中可以使用 Python 丰富的库生态系统（例如，用于数据科学、机器学习），并结合 Java 的性能和可伸缩性 [21]。Py4j 促进了多语言分布式系统的开发，在这些系统中，不同的组件可以使用最合适的语言来实现 [7]。它还为以 Python 为中心的数据科学家和以 Java 为中心的基础架构工程师之间架起了一座相对直接的桥梁 [5]。Py4j 处理回调的能力在异步通信和事件处理常见的分布式系统中尤为重要 [18]。

结论

总而言之，PySpark 和 PyFlink 都依赖 Py4j 的主要原因是 Py4j 提供了一个健壮且成熟的解决方案，能够让 Python 程序与 JVM 中的 Java 对象进行交互 [56]。其客户端-服务器架构和基于套接字的通信非常适合 Spark 和 Flink 的分布式特性 [13]。Py4j 支持回调和异常处理等关键功能，这对于 Python API 的功能至关重要 [13]。PySpark 的广泛采用及其活跃的开发使其成为 PyFlink 的可靠选择 [54]。最终，Py4j 在弥合用户友好的 Python API 和 PySpark 及 PyFlink 的高性能 Java/Scala 分布式计算引擎之间的差距方面发挥着基础性的作用。

Works cited

1. PySpark - The Examples Book, accessed April 21, 2025,

https://the-examples-book.com/tools/anvil/pyspark
2. What is PySpark? - Databricks, accessed April 21, 2025, https://www.databricks.com/glossary/pyspark
3. Pyspark Tutorial: Getting Started with Pyspark - DataCamp, accessed April 21, 2025, https://www.datacamp.com/tutorial/pyspark-tutorial-getting-started-with-pyspark
4. Apache Spark vs Flink, a detailed comparison - Macrometa, accessed April 21, 2025, https://www.macrometa.com/event-stream-processing/spark-vs-flink
5. A Hands-On Introduction to PyFlink - Decodable, accessed April 21, 2025, https://www.decodable.co/blog/a-hands-on-introduction-to-pyflink
6. Getting Started With PyFlink on Kubernetes - Decodable, accessed April 21, 2025, https://www.decodable.co/blog/getting-started-with-pyflink-on-kubernetes
7. PyFlink: Architecture and Applicable Business Scenarios - Alibaba Cloud Community, accessed April 21, 2025, https://www.alibabacloud.com/blog/597663
8. All You Need to Know About PyFlink - Ververica, accessed April 21, 2025, https://www.ververica.com/blog/all-you-need-to-know-about-pyflink
9. PyFlink — A deep dive into Flink's Python API - Quix, accessed April 21, 2025, https://quix.io/blog/pyflink-deep-dive
10. How to fix common issues when using Spark Structured Streaming with PySpark and Kafka, accessed April 21, 2025, https://quix.io/blog/how-to-fix-common-issues-spark-structured-streaming-pyspark-kafka
11. PySpark | Python API for Apache Spark - Domino Data Lab, accessed April 21, 2025, https://domino.ai/data-science-dictionary/pyspark
12. www.py4j.org, accessed April 21, 2025, https://www.py4j.org/#:~:text=Py4J%20enables%20Python%20programs%20running,through%20standard%20Python%20collection%20methods.
13. Welcome to Py4J — Py4J, accessed April 21, 2025, https://www.py4j.org/
14. Understanding the role of Py4J in Databricks - Perficient Blogs, accessed April 21, 2025, https://blogs.perficient.com/2024/03/10/understanding-the-role-of-py4j-in-databricks/
15. Py4j - MyRobotLab, accessed April 21, 2025, https://myrobotlab.org/service/Py4j
16. py4j · PyPI, accessed April 21, 2025, https://pypi.org/project/py4j/
17. Py4J enables Python programs to dynamically access arbitrary Java objects - GitHub, accessed April 21, 2025, https://github.com/py4j/py4j
18. 3. Advanced Topics - Py4J, accessed April 21, 2025, https://www.py4j.org/advanced_topics.html
19. PySpark and the JVM - introduction, part 1 on waitingforcode.com, accessed April 21, 2025, https://www.waitingforcode.com/pyspark/pyspark-jvm-introduction-1/read
20. Python, Spark and the JVM: An overview of the PySpark Runtime Architecture, accessed April 21, 2025, https://dev.to/steadbytes/python-spark-and-the-jvm-an-overview-of-the-pyspar

k-runtime-architecture-21gg
21. Is Jython dead? : r/learnpython - Reddit, accessed April 21, 2025, https://www.reddit.com/r/learnpython/comments/66r2l1/is_jython_dead/
22. 2. Getting Started with Py4J, accessed April 21, 2025, https://www.py4j.org/getting_started.html
23. java - Simplest example with py4J - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/22386399/simplest-example-with-py4j
24. How to call java from python using PY4J - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/20754129/how-to-call-java-from-python-using-py4j
25. 4.1. py4j.java_gateway — Py4J Main API, accessed April 21, 2025, https://www.py4j.org/py4j_java_gateway.html
26. Both ways, java <=> python, communication using py4j - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/44149878/both-ways-java-python-communication-using-py4j
27. py4j/py4j-python/src/py4j/protocol.py at master - GitHub, accessed April 21, 2025, https://github.com/py4j/py4j/blob/master/py4j-python/src/py4j/protocol.py
28. About Py4J, accessed April 21, 2025, https://www.py4j.org/about.html
29. Apache Spark with Python 101—Quick Start to PySpark (2025) - Chaos Genius, accessed April 21, 2025, https://www.chaosgenius.io/blog/apache-spark-with-python/
30. Understanding Apache Spark Architecture: Key Components - Encora, accessed April 21, 2025, https://insights.encora.com/insights/apache-spark-architecture
31. Python, Spark and the JVM: An overview of the PySpark Runtime Architecture - SteadBytes, accessed April 21, 2025, https://steadbytes.com/blog/pyspark-runtime-architecture/
32. Debugging PySpark — PySpark 3.5.5 documentation - Apache Spark, accessed April 21, 2025, https://spark.apache.org/docs/latest/api/python/development/debugging.html
33. Creating pyspark's spark context py4j java gateway object - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/66797382/creating-pysparks-spark-context-py4j-java-gateway-object
34. PySpark and the JVM - introduction, part 2 on waitingforcode.com, accessed April 21, 2025, https://www.waitingforcode.com/pyspark/pyspark-jvm-introduction-2/read
35. What is the entry point for its py4j gatewayServer in spark? - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/35774347/what-is-the-entry-point-for-its-py4j-gatewayserver-in-spark
36. Resolving py4j.protocol.Py4JJavaError in PyIceberg and PySpark | Orchestra, accessed April 21, 2025, https://www.getorchestra.io/guides/resolving-py4j-protocol-py4jjavaerror-in-pyic

eberg-and-pyspark
37. Getting Started with PyFlink: My Local Development Experience - - Armas.AI , accessed April 21, 2025, https://alejandroarmas.pages.dev/post/pyflink_debugging/
38. PyFlink: Introducing Python Support for UDFs in Flink's Table API, accessed April 21, 2025, https://flink.apache.org/2020/04/09/pyflink-introducing-python-support-for-udfs-in-flinks-table-api/
39. Everything You Need to Know about PyFlink - Alibaba Cloud Community, accessed April 21, 2025, https://www.alibabacloud.com/blog/everything-you-need-to-know-about-pyflink_599959
40. apache flink - What's wrong with my Pyflink setup that Python UDFs throw py4j exceptions?, accessed April 21, 2025, https://stackoverflow.com/questions/68015759/whats-wrong-with-my-pyflink-setup-that-python-udfs-throw-py4j-exceptions
41. JPype User Guide - Read the Docs, accessed April 21, 2025, https://jpype.readthedocs.io/en/latest/userguide.html
42. Calling Java from Python - LIACS Wiki, accessed April 21, 2025, https://rsewiki.liacs.nl/calling_java_from_python
43. jzy3d/pyzy3d: Python binding to Jzy3d using Py4j or Jython - GitHub, accessed April 21, 2025, https://github.com/jzy3d/pyzy3d
44. Referring to Jython scripts in Py4J scripts - Google Groups, accessed April 21, 2025, https://groups.google.com/a/py4j.org/g/py4j/c/I4I9DWF52u0
45. Jython - is it faster to use Python data structures or Java ones? - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/11178243/jython-is-it-faster-to-use-python-data-structures-or-java-ones
46. A brief overview of Python-Java bridges in 2020 - talvi.net, accessed April 21, 2025, https://talvi.net/posts/a-brief-overview-of-python-java-bridges-in-2020.html
47. Best way to combine Python and Java? - Reddit, accessed April 21, 2025, https://www.reddit.com/r/java/comments/ygnh2q/best_way_to_combine_python_and_java/
48. Interop with Java : r/Python - Reddit, accessed April 21, 2025, https://www.reddit.com/r/Python/comments/9ndile/interop_with_java/
49. Python for Java APIs - NXCALS Documentation, accessed April 21, 2025, https://nxcals-docs.web.cern.ch/1.4.2/user-guide/data-access/py4j-jpype-installation/
50. Py4J has bigger overhead than Jython and JPype - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/18484879/py4j-has-bigger-overhead-than-jython-and-jpype
51. PyJnius vs JPype · Issue #551 - GitHub, accessed April 21, 2025, https://github.com/kivy/pyjnius/issues/551

52. What advantages does this have over JPype? · Issue #2 - GitHub, accessed April 21, 2025, https://github.com/karpierz/jtypes.jpype/issues/2
53. What is the best way to run the same java function repeatedly in python? - Stack Overflow, accessed April 21, 2025, https://stackoverflow.com/questions/49704105/what-is-the-best-way-to-run-the-same-java-function-repeatedly-in-python
54. Switching from pyjnius to py4j #833 - castorini/anserini - GitHub, accessed April 21, 2025, https://github.com/castorini/anserini/issues/833
55. Top 10 Examples of py4j code in Python - CloudDefense.AI, accessed April 21, 2025, https://www.clouddefense.ai/code/python/example/py4j
56. Empirical Study Between Compiled, Interpreted, and Dynamic P... - KnE Open, accessed April 21, 2025, https://kneopen.com/KnE-Engineering/article/view/3650/