



US011733984B1

(12) **United States Patent**
Kumar et al.

(10) **Patent No.:** US 11,733,984 B1
(45) **Date of Patent:** Aug. 22, 2023

(54) **GENERATING A PLATFORM-AGNOSTIC DATA PIPELINE VIA A LOW CODE TRANSFORMATION LAYER SYSTEMS AND METHODS**

(71) Applicant: **Citibank, N.A.**, New York, NY (US)

(72) Inventors: **Vaibhav Kumar**, Monroe, NJ (US); **Ashutosh Pandey**, Brandon, FL (US); **Tanujit Ghosh**, Mississauga (CA)

(73) Assignee: **Citibank, N.A.**, New York, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **18/124,543**

(22) Filed: **Mar. 21, 2023**

(51) **Int. Cl.**
G06F 8/41 (2018.01)
G06F 21/57 (2013.01)

(52) **U.S. Cl.**
CPC *G06F 8/4452* (2013.01); *G06F 21/577* (2013.01)

(58) **Field of Classification Search**

None

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

11,188,525 B2 * 11/2021 Athavale G06F 16/2458

11,226,813 B2 * 1/2022 Gungabeesoon G06F 8/60

2018/0052861 A1	2/2018	Seetharaman et al.
2018/0342324 A1 *	11/2018	Cha
2021/0034581 A1	2/2021	Boven et al.
2021/0064742 A1 *	3/2021	Panigrahi
2021/0304021 A1 *	9/2021	Puri
2022/0121981 A1	4/2022	Chaudhuri et al.
2022/0245115 A1	8/2022	Kulkarni et al.
2022/0398186 A1 *	12/2022	Mudumba
G06F 8/30		

* cited by examiner

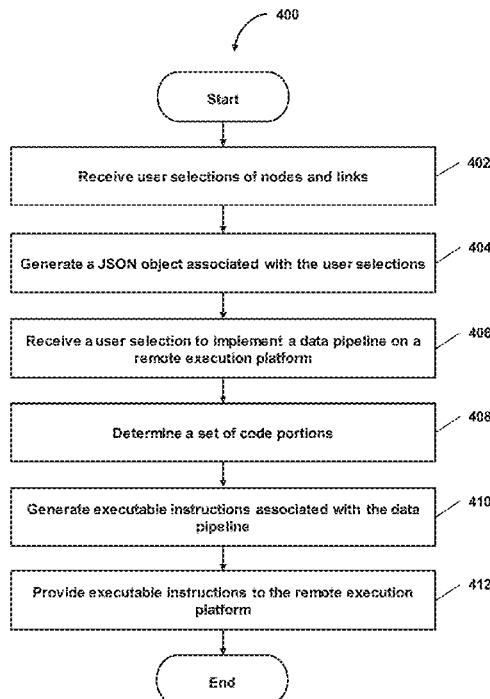
Primary Examiner — Marina Lee

(74) *Attorney, Agent, or Firm* — Perkins Coie LLP

(57) **ABSTRACT**

Systems and methods for generating a platform-agnostic data pipeline via a low code transformation layer are disclosed. The system receives one or more user selections of (i) nodes and (ii) links linking the nodes, indicating a data pipeline architecture of transfer/management/flow of data via a GUI. In response to receiving a user selection to implement the data pipeline, the system automatically identifies/generates a set of code portions, based on one or more software objects (e.g., JSON objects) associated with the user selections indicating the data pipeline architecture. The system then identifies a platform identifier associated with a remote server and generates a set of executable instructions (e.g., a script, executable program, or other file) associated with the data pipeline architecture by using a transformation component. The system then provides the executable instructions to the remote server to host the data pipeline.

20 Claims, 5 Drawing Sheets



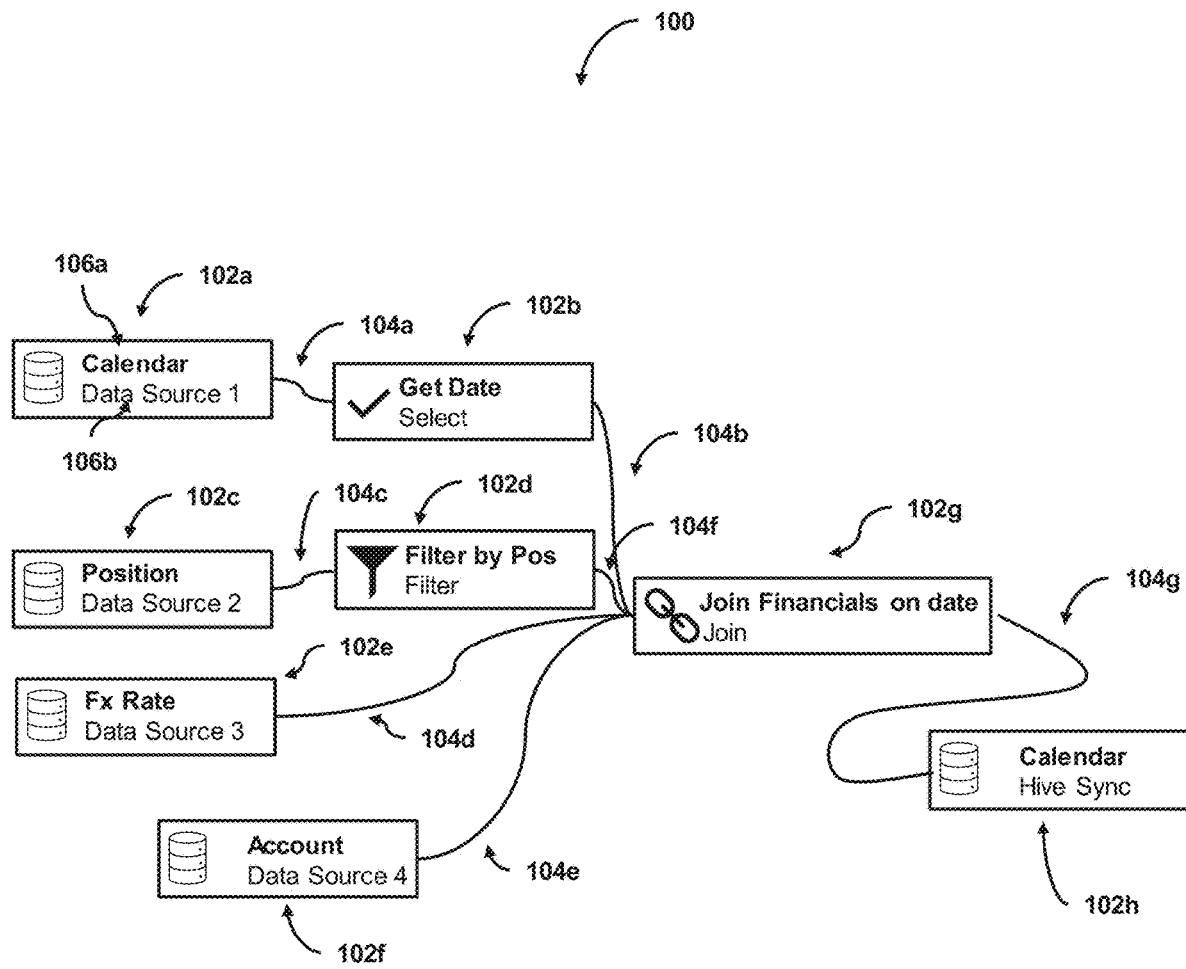


FIG. 1

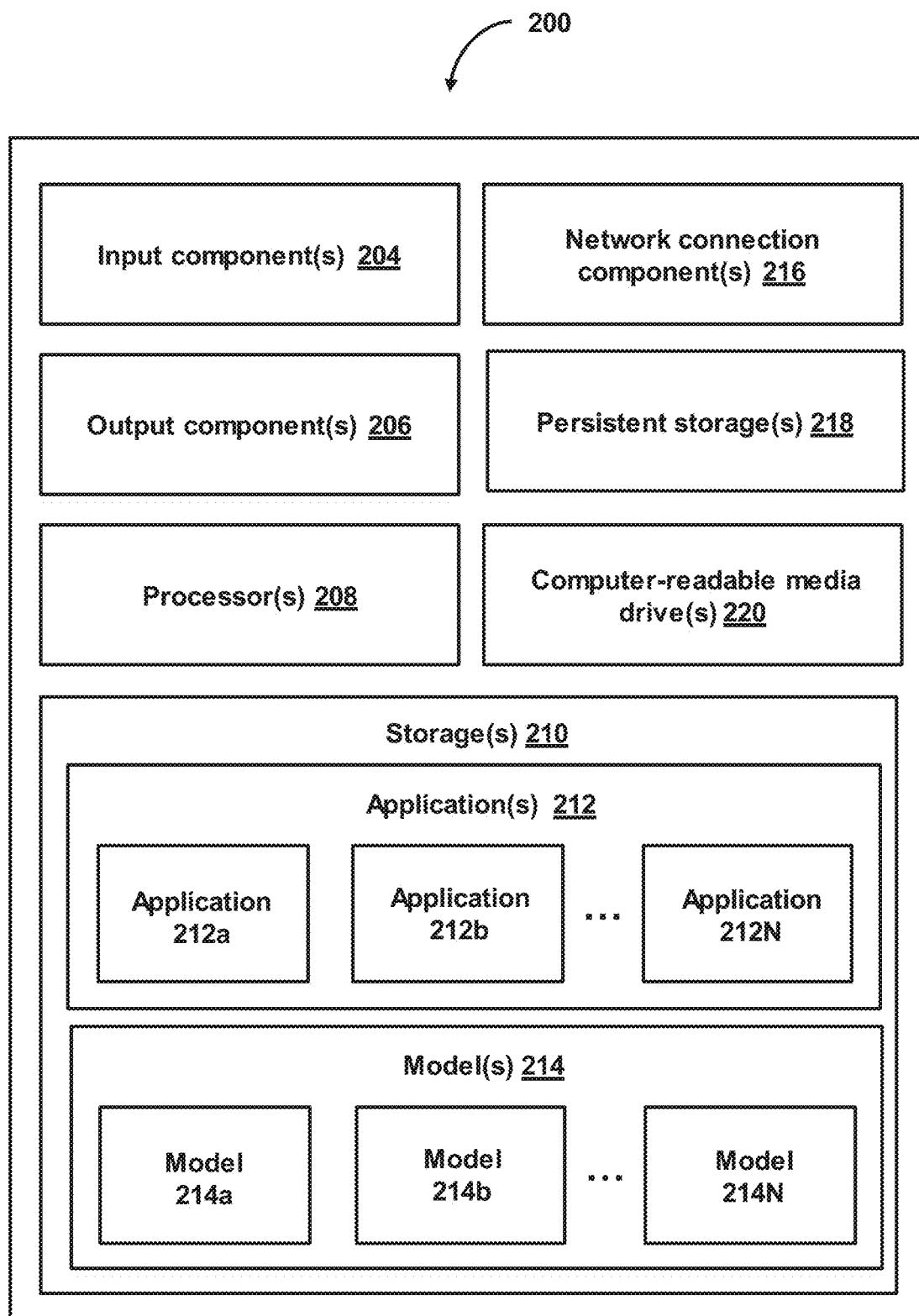
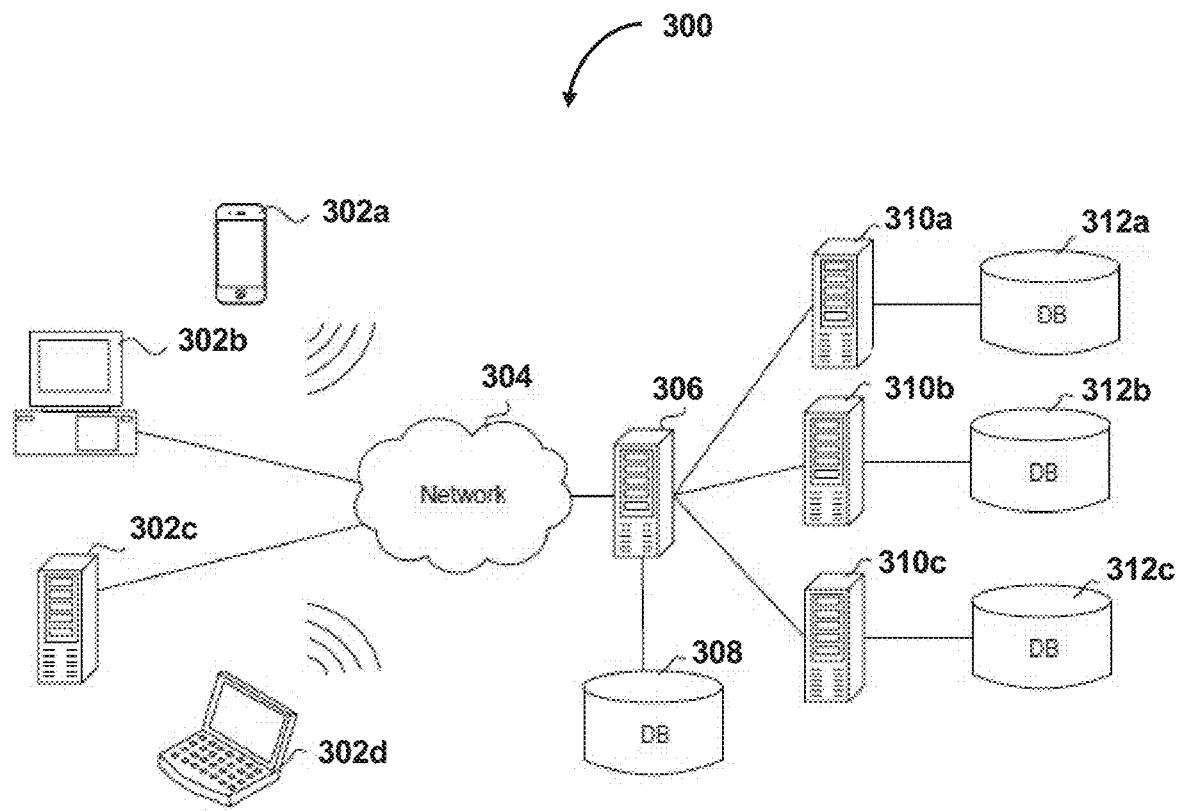


FIG. 2

**FIG. 3**

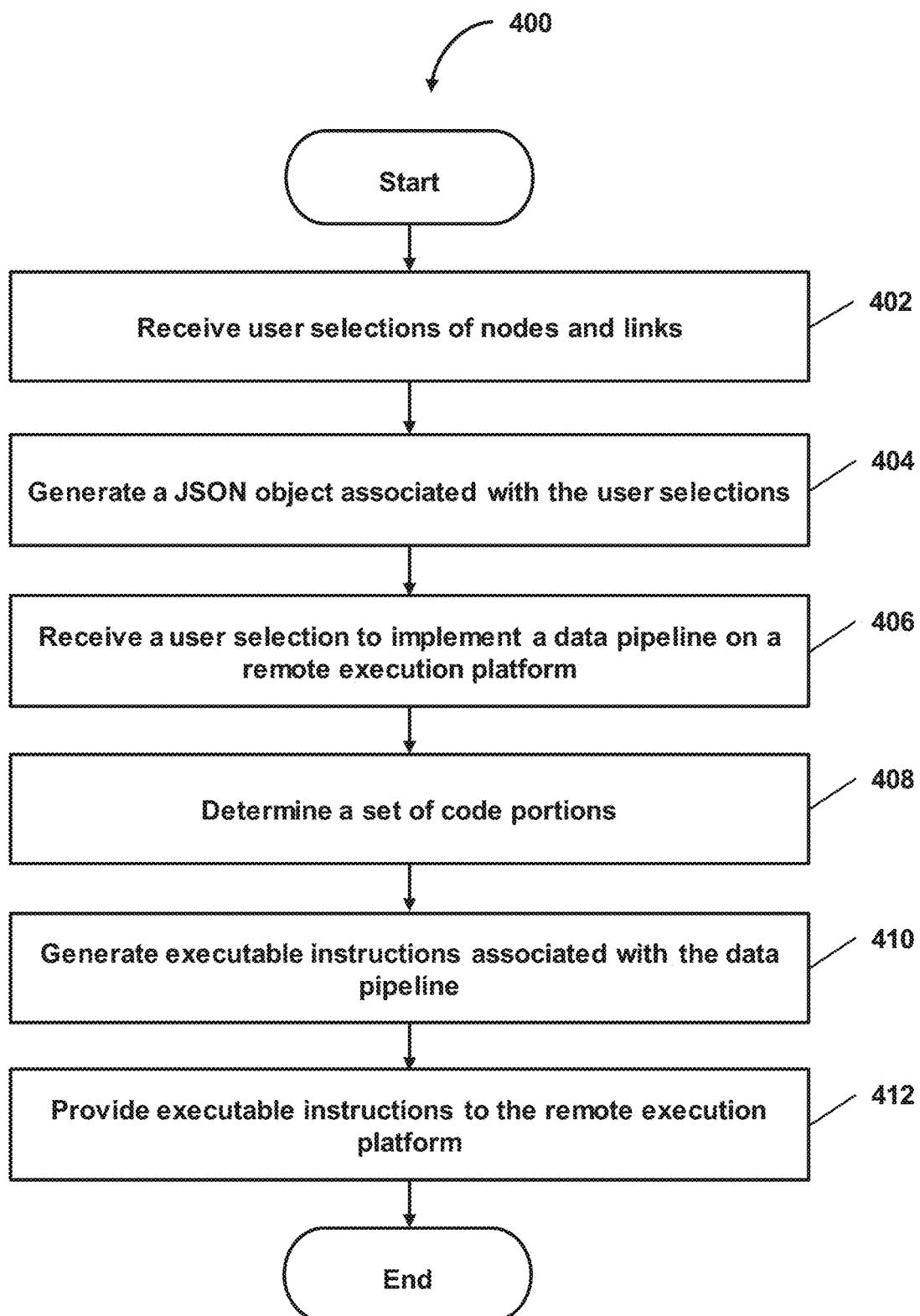
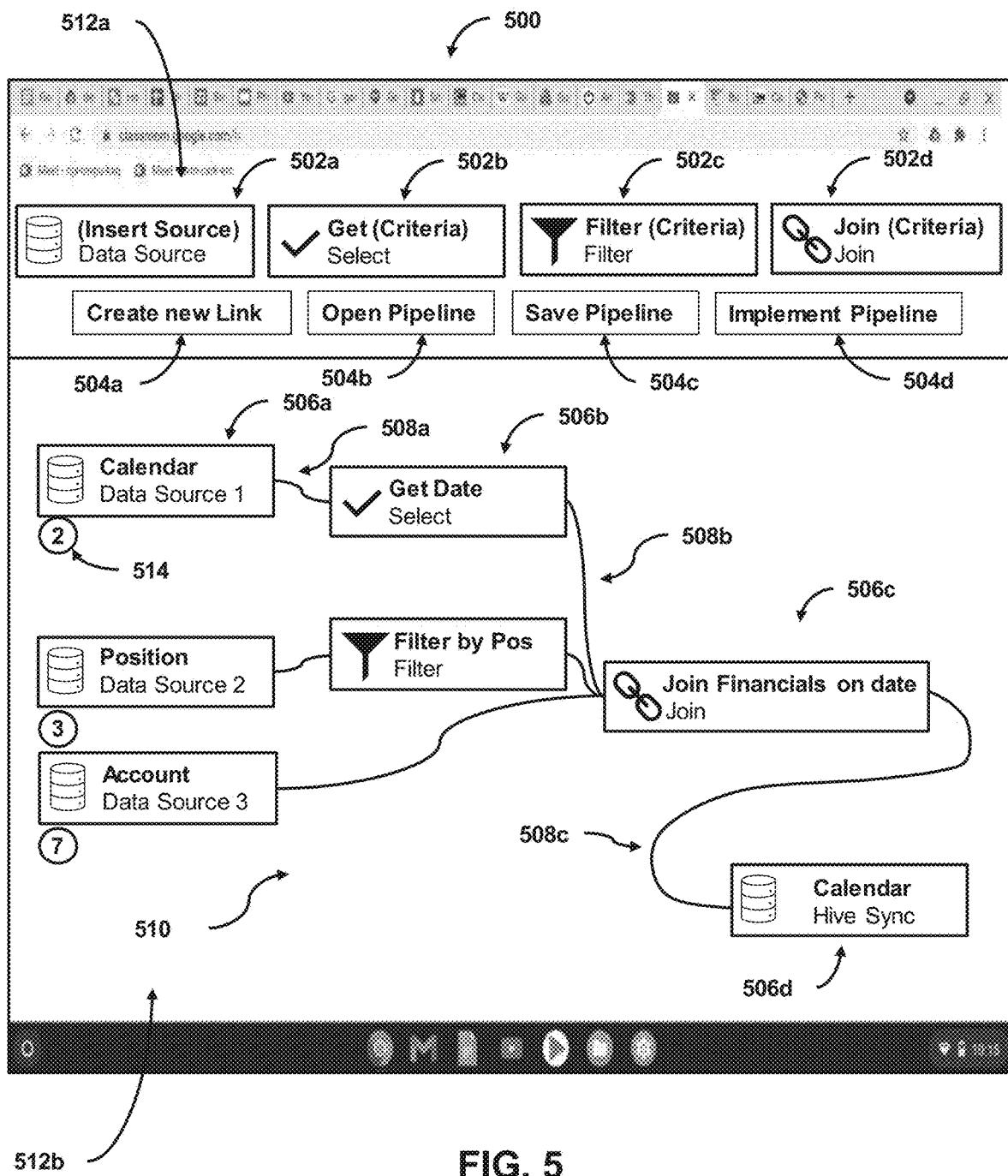


FIG. 4



**GENERATING A PLATFORM-AGNOSTIC
DATA PIPELINE VIA A LOW CODE
TRANSFORMATION LAYER SYSTEMS AND
METHODS**

BACKGROUND

Creating robust and useful data pipelines is often a task spearheaded by computer scientists and data analysts due to the sheer complexity and knowledge required to create data pipelines. For instance, a data pipeline may represent a chain of data processing elements that can transform and move raw data from one or more sources to a destination. For example, when creating a data pipeline, a user must understand the intricacies involved with the type of data, how to filter the data, how to combine data of differing sources, and ultimately how to export the data in an easily understood format. Moreover, creating a data pipeline takes an exorbitant amount of time which may lead to useful data currently being provided by data sources to be missed or unconsidered when creating the data pipeline. Furthermore, end-users (e.g., business professionals or other non-specialized/non-technical users) have no knowledge/mechanism for creating data pipelines due to the complexity involved with creating data pipelines. Such users may understand the need for a given data set but are otherwise unable to obtain the desired data in a usable format due to lack of knowledge of how to create or set up a data pipeline required for obtaining the data set. Moreover, data pipelines may be configured for a given platform only (e.g., computing system, operating system, processor, etc.) and may not be scalable to other platforms, further increasing the difficulty when creating/using a data pipeline.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is an illustrative platform-agnostic data pipeline via a low code transformation layer, in accordance with some implementations of the present technology.

FIG. 2 is a block diagram showing some of the components typically incorporated in at least some of the computer systems and other devices on which the disclosed system operates in accordance with some implementations of the present technology.

FIG. 3 is a system diagram illustrating an example of a computing environment in which the disclosed system operates in some implementations of the present technology.

FIG. 4 is a flow diagram illustrating a process of generating a platform-agnostic data pipeline via a low code transformation layer, in accordance with some implementations of the present technology.

FIG. 5 shows an illustrative representation of a Graphical User Interface (GUI) for generating a platform-agnostic data pipeline via a low code transformation layer, in accordance with some implementations of the present technology.

In the drawings, some components and/or operations can be separated into different blocks or combined into a single block for discussion of some of the implementations of the present technology. Moreover, while the technology is amenable to various modifications and alternative forms, specific implementations have been shown by way of example in the drawings and are described in detail below. The intention, however, is not to limit the technology to the specific implementations described. On the contrary, the technology is intended to cover all modifications, equiva-

lents, and alternatives falling within the scope of the technology as defined by the appended claims.

DETAILED DESCRIPTION

Given the complexity and specialized knowledge required for creating robust data pipelines, end-users are at a loss when attempting to obtain the necessary data required to analyze a given trend. For instance, business professionals 10 are not trained nor do they possess the specialized knowledge to create a data pipeline from scratch, although the business professionals may know exactly the type of data they need to analyze. Existing solutions currently require that a business professional (e.g., an analyst) speak to a translator who may develop a strategic plan on which data sources are to be used and how one may alter the data (e.g., by filtering, combining, syncing, fitting, or otherwise operate on the data) to provide an easily understood format of the data to be used for analysis purposes. However, this still 15 does not solve the problem of creating the data pipeline as the translator must communicate the plan with a computer/data engineer to physically create, design, and interconnect the computing architectures required to deliver the data in the requested format. This may lead to an increased amount of wasted time, money, and disconnect as the amount of people required to implement the data pipeline is increased. In addition, the resulting data pipeline may not adequately 20 conform to the original requirements outlined by the business professional, leading to additional effort and resource 25 usage to correct the issues.

Another common issue faced when creating data pipelines is determining which platform is best suited for the data pipeline. For instance, as conventional software applications (e.g., a data pipeline, legacy software, etc.) may be specific 30 to a particular platform due to the compilers used to compile the source code associated with the data pipeline, the computer/data engineers must carefully select which computing language to write the source code in and which platform the pipeline will execute on. This leads to scalability issues as 35 one pipeline may not execute properly (if at all) on a different platform as the new processor may be of a different architecture (e.g., instruction set) that is incompatible with a generated executable file of the original pipeline. Although 40 computer engineers may recompile the source code on a different compiler, such executable files are not easily shared. For example, when a large amount of users are using the pipeline on a given platform, when the platform changes, the computer engineers must recompile the original source code, distribute the generated executable file associated with 45 the recompiled source code (e.g., over the Internet or other computing networks), and finally reinstall the updated executable file. This leads to an increased amount of wasted computer processing resources, computer memory resources, and an increased amount of network traffic that 50 otherwise may be used for other functions/operations.

Existing systems currently are designed for computer savvy computer/data engineers (e.g., users trained/skilled at generating software code) to create data pipelines that are 55 specific to a given platform. However, without a mechanism to enable non-computer savvy users to create data pipelines, such users are at a loss for creating or implementing valuable data pipelines to obtain and analyze crucial data. Moreover, as the computer engineers may not fully understand one or more requirements an analyst needs to analyze the data 60 appropriately, creating the data pipeline may be prone to numerous errors causing the computer engineers to go back, edit, and modify the data pipeline to fit the analysts' needs—

thereby increasing the time at which these data pipelines are created while wasting valuable computing processing and memory resources at the same time. Lastly, as these existing systems create data pipelines custom tailored to a specific platform, users of other platforms cannot access the data pipeline or receive updates without waiting for a newly created executable file of the data pipeline to be implemented, further causing user frustration and increasing the amount of network traffic when updates or the executable file is shared.

Thus, in light of these and other problems with conventional solutions and systems, there is a need for providing a simplified user interface that can enable users, such as business professionals, to create data pipelines without needing specialized training to create the necessary data pipeline. There is further a need to enable quick generation of these data pipelines to reduce the amount of time spent creating, adjusting, and modifying a data pipeline. Moreover, there is a need to create platform-agnostic data pipelines to enable data pipelines to be implemented on a variety of differing platforms, to provide updates to such data pipelines in a scalable and efficient manner, and to quickly share an implemented pipeline amongst users of different platforms.

The inventors have developed a system for generating a platform-agnostic data pipeline via a low code transformation layer. For instance, the inventors have developed a low code transformation layer that uses one or more of an improved graphical user interface (GUI), a template, and predetermined code portions to generate a platform-agnostic data pipeline. End-users, such as business professionals, can use the GUI to easily select one or more nodes representing data pipelines operations and connect the nodes via one or more links. Using the nodes and the links, the system may generate a template in a first computing language in which the low code transformation layer may determine predetermined code portions that correspond to the one or more nodes and links to generate executable instructions in a second computing language corresponding to a remote execution platform (e.g., to which the data pipeline is to be executed, hosted, or ran on). By using the improved user interface, users without specialized knowledge on how to implement a data pipeline are enabled to quickly create and develop their own data pipelines that are customizable and tailored to their needs. Furthermore, by using the low code transformation layer, the created data pipeline may be platform-agnostic, meaning, that the data pipeline can be executed on any given platform irrespective of specific computing hardware (e.g., processors) or other computing-platform-related infrastructure. By doing so, the system may reduce the amount of computer processing and memory resources conventionally required to create, share, and provide updates to the data pipeline.

In various implementations, the methods and systems described herein can generate a platform-agnostic data pipeline via a low code transformation layer. For example, the system receives user selection of (i) nodes and (ii) a set of links linking the nodes via a graphical user interface (GUI). The user selections can indicate an architecture of a data pipeline. The system can generate, using the user selections, a JSON object (e.g., or other software object/file) associated with the user selections indicating the data pipeline architecture. In response to receiving a user selection to implement the data pipeline (e.g., to be hosted on a remote execution platform), the system can determine, using the JSON object, a set of code portions. Each code portion of the set of code portions can correspond to a respective combi-

nation of at least (i) a first and a second node of the user selections and (ii) a link linking the first and second nodes. The system generates executable instructions (or other computer readable instructions) associated with the data pipeline architecture by using a transformation component configured to process the set of code portions with respect to the remote execution platform using the JSON object. The system then provides the executable instructions to the remote execution platform to host the data pipeline.

In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of implantations of the present technology. It will be apparent, however, to one skilled in the art that implementation of the present technology can be practiced without some of these specific details.

The phrases "in some implementations," "in several implementations," "according to some implementations," "in the implementations shown," "in other implementations," and the like generally mean the specific feature, structure, or characteristic following the phrase is included in at least one implementation of the present technology and can be included in more than one implementation. In addition, such phrases do not necessarily refer to the same implementations or different implementations.

System Overview

FIG. 1 shows an illustrative platform-agnostic data pipeline via a low code transformation layer, in accordance with some implementations of the present technology. For example, a user (e.g., an analyst, a computer engineer, a data engineer, a corporate executive, or other user) can create a platform-agnostic data pipeline **100**. For example, the platform-agnostic data pipeline may include nodes **102a-102h** (or as collectively referred to as nodes **102**) and links **104a-104g** (or as collectively referred to links **104**). Each node **102** can be associated with a data source or an operation. As an example, a data source can be any database, table, data structure, or other data source configured to store data. As another example, an operation can be any operation that performs an action (e.g., on data, associated with data, or with respect to data). For instance, an operation may be a filter operation, a transform operation, a join operation, a select operation, a remove operation, a group operation, a load to database operation, drop column operation, drop row operation, add column operation, add row operation, sort operation, format operation, or other operation that may be performed on or performed with data.

Each node or operation can include or be provided with one or more user selectable commands or user provided instructions. For example, a filter operation may filter data by a position or other criteria (e.g., as inputted by the user). In some implementations, the user selectable commands or user provided instructions may be default commands or instructions (e.g., not requiring a user to select or provide instructions). For example, a join operation may be associated with default commands to append rows from one data source to another data source. Alternatively, the join operation may be associated with user selectable commands (e.g., indicating to join only certain rows/columns of one data source with certain rows/columns of another data source). Such user provided commands/instructions may be provided by the user (e.g., selecting one or more commands/instructions from a list, drop down menu, or other interface, providing one or more commands/instructions by typing in the commands/instructions, performing one or more gestures, or otherwise providing the commands/instructions). In

some implementations, the commands or instructions may further indicate a set of criteria, as will be described below.

A filter operation can filter data according to a date, time, position, value, a format, a length, a size, or other value associated with the data. A transform operation can transform data into a given format, metric, size, measurement, currency, range, or other transformation criteria associated with the data. A join operation can join data of a set of data sources together, such as joining rows/columns of a data source(s), appending data to a data structure, or other joining operation with respect to a set of data sources. A select operation can select, obtain, get, or retrieve data with respect to a given criteria (e.g., select data based on a date, select data based on a format, select data based on a type, etc.). A remove operation may remove data from a given data source/data structure, such as removing rows/columns, removing values, removing headers, removing nodes of a tree data structure, removing links of a tree data structure, or other data removing operation. A group operation may group data together, such as grouping data based on a type, size, range, date, time, value, or other data grouping criteria. A load to database operation may upload data to a given data source, database, data structure, update a database/data structure, or other database loading operation. A drop column operation may remove, hide, or delete a given column (e.g., in a table data structure). A drop row operation may remove, hide, or delete a given row (e.g., in a table data structure). An add column operation may add, append, update, or join a column (e.g., in a table data structure). An add row operation may add, append, update, or join a row (e.g., in a table data structure). A sort operation may sort information based on a time, date, value, size, metric, or other data sorting criteria. A format operation may format data of a data source/data structure such as formatting data to a particular size, value, length, date, time, or other data formatting criteria. A conversion operation may convert data from one format to another format, such as converting data from one currency to another currency, one converting data from one metric to another metric, or other conversion criteria.

As shown in FIG. 1, each node can be labeled with a set of labels. For example, first node 102a may include first label 106a and second label 106b. Each label can indicate a reference value and a type value. For example, first label 106a indicates a reference value of “Calendar” indicating that the data source is a calendar data source, and second label 106b indicates a type value indicating that the node is associated with a “Data Source” (or data base type). In this way, when creating a data pipeline, users are provided with clear labels indicating an identifier associated with the node (e.g., a label associated with a given operation or data source to which data is modified, changed, altered, or obtained from) and a type identifier indicating what type of node is selected (e.g., a data source node, an operation node, or other node), thereby improving the user experience via an improved user interface.

The platform-agnostic data pipeline 100 can also include links 104. For example, the links may link together a set of nodes. For instance, first link 104a links first node 102a and second node 102b. In some implementations, each link 104 may be a directional link (e.g., indicating a data flow between one or more nodes). In other implementations, each link 104 may be a bi-directional link (e.g., indicating that data may flow back and forth between each node). In yet other implementations, the platform-agnostic data pipeline

100 may include a mixture of both directional and bi-directional links, in accordance with some implementations of the present technology.

Linking a set of nodes together may be associated with 5 “connecting” or “supplementing” data associated with one node to data associated with another node (or an operation to another operation, or other combination). For example, to provide users with an easy-to-use user interface when creating data pipelines, as opposed to existing systems that 10 require computer engineers and data engineers to manually code each interaction between a data source and an operation, users may quickly and easily link nodes together to forego the complex knowledge and specialized skills required to write code to implement a data pipeline architecture.

Links 104 can represent a path connecting two nodes together. For instance, as nodes may be associated with a data source, an operation, or an output (e.g., a visualization of data, a data structure, formatted data, etc.), when a node 20 is linked to another node, data may be passed between the node(s). As an example, first node 102a may be linked via first link 104a to second node 102b. Where first node 102a represents a data source (e.g., a database) and second node 102b represents an operation node (e.g., select operation). 25 As first node 102a and second node 102b are linked together, and second node 102b is an operation node (e.g., select operation node), second node 102b may retrieve data that may be associated with first node 102a. That is, in this example, first node 102a may be associated with a data 30 source including dates and second node 102b may be configured to retrieve the dates associated with first node 102a via first link 104a. It should be noted that other combinations of nodes linked together may exist, in accordance with some implementations of the present technology. 35 As such, links 104 may operationally connect nodes together to quickly and easily generate a flow of data between one or more nodes. In this way, the system may generate data pipelines without requiring users to be proficient in data pipeline architecture creation, thereby improving the user 40 experience.

Suitable Computing Environments

FIG. 2 is a block diagram showing some of the components typically incorporated in at least some of the computer systems and other devices on which the disclosed system operates. In various implementations, these computer systems and other device(s) 200 can include server computer systems, desktop computer systems, laptop computer systems, netbooks, mobile phones, personal digital assistants, televisions, cameras, automobile computers, electronic media players, web services, mobile devices, watches, wearables, glasses, smartphones, tablets, smart displays, virtual reality devices, augmented reality devices, etc. In various implementations, the computer systems and devices include zero or more of each of the following: input components 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985 990 995 1000 1005 1010 1015 1020 1025 1030 1035 1040 1045 1050 1055 1060 1065 1070 1075 1080 1085 1090 1095 1100 1105 1110 1115 1120 1125 1130 1135 1140 1145 1150 1155 1160 1165 1170 1175 1180 1185 1190 1195 1200 1205 1210 1215 1220 1225 1230 1235 1240 1245 1250 1255 1260 1265 1270 1275 1280 1285 1290 1295 1300 1305 1310 1315 1320 1325 1330 1335 1340 1345 1350 1355 1360 1365 1370 1375 1380 1385 1390 1395 1400 1405 1410 1415 1420 1425 1430 1435 1440 1445 1450 1455 1460 1465 1470 1475 1480 1485 1490 1495 1500 1505 1510 1515 1520 1525 1530 1535 1540 1545 1550 1555 1560 1565 1570 1575 1580 1585 1590 1595 1600 1605 1610 1615 1620 1625 1630 1635 1640 1645 1650 1655 1660 1665 1670 1675 1680 1685 1690 1695 1700 1705 1710 1715 1720 1725 1730 1735 1740 1745 1750 1755 1760 1765 1770 1775 1780 1785 1790 1795 1800 1805 1810 1815 1820 1825 1830 1835 1840 1845 1850 1855 1860 1865 1870 1875 1880 1885 1890 1895 1900 1905 1910 1915 1920 1925 1930 1935 1940 1945 1950 1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020 2025 2030 2035 2040 2045 2050 2055 2060 2065 2070 2075 2080 2085 2090 2095 2100 2105 2110 2115 2120 2125 2130 2135 2140 2145 2150 2155 2160 2165 2170 2175 2180 2185 2190 2195 2200 2205 2210 2215 2220 2225 2230 2235 2240 2245 2250 2255 2260 2265 2270 2275 2280 2285 2290 2295 2300 2305 2310 2315 2320 2325 2330 2335 2340 2345 2350 2355 2360 2365 2370 2375 2380 2385 2390 2395 2400 2405 2410 2415 2420 2425 2430 2435 2440 2445 2450 2455 2460 2465 2470 2475 2480 2485 2490 2495 2500 2505 2510 2515 2520 2525 2530 2535 2540 2545 2550 2555 2560 2565 2570 2575 2580 2585 2590 2595 2600 2605 2610 2615 2620 2625 2630 2635 2640 2645 2650 2655 2660 2665 2670 2675 2680 2685 2690 2695 2700 2705 2710 2715 2720 2725 2730 2735 2740 2745 2750 2755 2760 2765 2770 2775 2780 2785 2790 2795 2800 2805 2810 2815 2820 2825 2830 2835 2840 2845 2850 2855 2860 2865 2870 2875 2880 2885 2890 2895 2900 2905 2910 2915 2920 2925 2930 2935 2940 2945 2950 2955 2960 2965 2970 2975 2980 2985 2990 2995 3000 3005 3010 3015 3020 3025 3030 3035 3040 3045 3050 3055 3060 3065 3070 3075 3080 3085 3090 3095 3100 3105 3110 3115 3120 3125 3130 3135 3140 3145 3150 3155 3160 3165 3170 3175 3180 3185 3190 3195 3200 3205 3210 3215 3220 3225 3230 3235 3240 3245 3250 3255 3260 3265 3270 3275 3280 3285 3290 3295 3300 3305 3310 3315 3320 3325 3330 3335 3340 3345 3350 3355 3360 3365 3370 3375 3380 3385 3390 3395 3400 3405 3410 3415 3420 3425 3430 3435 3440 3445 3450 3455 3460 3465 3470 3475 3480 3485 3490 3495 3500 3505 3510 3515 3520 3525 3530 3535 3540 3545 3550 3555 3560 3565 3570 3575 3580 3585 3590 3595 3600 3605 3610 3615 3620 3625 3630 3635 3640 3645 3650 3655 3660 3665 3670 3675 3680 3685 3690 3695 3700 3705 3710 3715 3720 3725 3730 3735 3740 3745 3750 3755 3760 3765 3770 3775 3780 3785 3790 3795 3800 3805 3810 3815 3820 3825 3830 3835 3840 3845 3850 3855 3860 3865 3870 3875 3880 3885 3890 3895 3900 3905 3910 3915 3920 3925 3930 3935 3940 3945 3950 3955 3960 3965 3970 3975 3980 3985 3990 3995 4000 4005 4010 4015 4020 4025 4030 4035 4040 4045 4050 4055 4060 4065 4070 4075 4080 4085 4090 4095 4100 4105 4110 4115 4120 4125 4130 4135 4140 4145 4150 4155 4160 4165 4170 4175 4180 4185 4190 4195 4200 4205 4210 4215 4220 4225 4230 4235 4240 4245 4250 4255 4260 4265 4270 4275 4280 4285 4290 4295 4300 4305 4310 4315 4320 4325 4330 4335 4340 4345 4350 4355 4360 4365 4370 4375 4380 4385 4390 4395 4400 4405 4410 4415 4420 4425 4430 4435 4440 4445 4450 4455 4460 4465 4470 4475 4480 4485 4490 4495 4500 4505 4510 4515 4520 4525 4530 4535 4540 4545 4550 4555 4560 4565 4570 4575 4580 4585 4590 4595 4600 4605 4610 4615 4620 4625 4630 4635 4640 4645 4650 4655 4660 4665 4670 4675 4680 4685 4690 4695 4700 4705 4710 4715 4720 4725 4730 4735 4740 4745 4750 4755 4760 4765 4770 4775 4780 4785 4790 4795 4800 4805 4810 4815 4820 4825 4830 4835 4840 4845 4850 4855 4860 4865 4870 4875 4880 4885 4890 4895 4900 4905 4910 4915 4920 4925 4930 4935 4940 4945 4950 4955 4960 4965 4970 4975 4980 4985 4990 4995 5000 5005 5010 5015 5020 5025 5030 5035 5040 5045 5050 5055 5060 5065 5070 5075 5080 5085 5090 5095 5100 5105 5110 5115 5120 5125 5130 5135 5140 5145 5150 5155 5160 5165 5170 5175 5180 5185 5190 5195 5200 5205 5210 5215 5220 5225 5230 5235 5240 5245 5250 5255 5260 5265 5270 5275 5280 5285 5290 5295 5300 5305 5310 5315 5320 5325 5330 5335 5340 5345 5350 5355 5360 5365 5370 5375 5380 5385 5390 5395 5400 5405 5410 5415 5420 5425 5430 5435 5440 5445 5450 5455 5460 5465 5470 5475 5480 5485 5490 5495 5500 5505 5510 5515 5520 5525 5530 5535 5540 5545 5550 5555 5560 5565 5570 5575 5580 5585 5590 5595 5600 5605 5610 5615 5620 5625 5630 5635 5640 5645 5650 5655 5660 5665 5670 5675 5680 5685 5690 5695 5700 5705 5710 5715 5720 5725 5730 5735 5740 5745 5750 5755 5760 5765 5770 5775 5780 5785 5790 5795 5800 5805 5810 5815 5820 5825 5830 5835 5840 5845 5850 5855 5860 5865 5870 5875 5880 5885 5890 5895 5900 5905 5910 5915 5920 5925 5930 5935 5940 5945 5950 5955 5960 5965 5970 5975 5980 5985 5990 5995 6000 6005 6010 6015 6020 6025 6030 6035 6040 6045 6050 6055 6060 6065 6070 6075 6080 6085 6090 6095 6100 6105 6110 6115 6120 6125 6130 6135 6140 6145 6150 6155 6160 6165 6170 6175 6180 6185 6190 6195 6200 6205 6210 6215 6220 6225 6230 6235 6240 6245 6250 6255 6260 6265 6270 6275 6280 6285 6290 6295 6300 6305 6310 6315 6320 6325 6330 6335 6340 6345 6350 6355 6360 6365 6370 6375 6380 6385 6390 6395 6400 6405 6410 6415 6420 6425 6430 6435 6440 6445 6450 6455 6460 6465 6470 6475 6480 6485 6490 6495 6500 6505 6510 6515 6520 6525 6530 6535 6540 6545 6550 6555 6560 6565 6570 6575 6580 6585 6590 6595 6600 6605 6610 6615 6620 6625 6630 6635 6640 6645 6650 6655 6660 6665 6670 6675 6680 6685 6690 6695 6700 6705 6710 6715 6720 6725 6730 6735 6740 6745 6750 6755 6760 6765 6770 6775 6780 6785 6790 6795 6800 6805 6810 6815 6820 6825 6830 6835 6840 6845 6850 6855 6860 6865 6870 6875 6880 6885 6890 6895 6900 6905 6910 6915 6920 6925 6930 6935 6940 6945 6950 6955 6960 6965 6970 6975 6980 6985 6990 6995 7000 7005 7010 7015 7020 7025 7030 7035 7040 7045 7050 7055 7060 7065 7070 7075 7080 7085 7090 7095 7100 7105 7110 7115 7120 7125 7130 7135 7140 7145 7150 7155 7160 7165 7170 7175 7180 7185 7190 7195 7200 7205 7210 7215 7220 7225 7230 7235 7240 7245 7250 7255 7260 7265 7270 7275 7280 7285 7290 7295 7300 7305 7310 7315 7320 7325 7330 7335 7340 7345 7350 7355 7360 7365 7370 7375 7380 7385 7390 7395 7400 7405 7410 7415 7420 7425 7430 7435 7440 7445 7450 7455 7460 7465 7470 7475 7480 7485 7490 7495 7500 7505 7510 7515 7520 7525 7530 7535 7540 7545 7550 7555 7560 7565 7570 7575 7580 7585 7590 7595 7600 7605 7610 7615 7620 7625 7630 7635 7640 7645 7650 7655 7660 7665 7670 7675 7680 7685 7690 7695 7700 7705 7710 7715 7720 7725 7730 7735 7740 7745 7750 7755 7760 7765 7770 7775 7780 7785 7790 7795 7800 7805 7810 7815 7820 7825 7830 7835 7840 7845 7850 7855 7860 7865 7870 7875 7880 7885 7890 7895 7900 7905 7910 7915 7920 7925 7930 7935 7940 7945 7950 7955 7960 7965 7970 7975 7980 7985 7990 7995 8000 8005 8010 8015 8020 8025 8030 8035 8040 8045 8050 8055 8060 8065 8070 8075 8080 8085 8090 8095 8100 8105 8110 8115 8120 8125 8130 8135 8140 8145 8150 8155 8160 8165 8170 8175 8180 8185 8190 8195 8200 8205 8210 8215 8220 8225 8230 8235 8240 8245 8250 8255 8260 8265 8270 8275 8280 8285 8290 8295 8300 8305 8310 8315 8320 8325 8330 8335 8340 8345 8350 8355 8360 8365 8370 8375 8380 8385 8390 8395 8400 8405 8410 8415 8420 8425 8430 8435 8440 8445 8450 8455 8460 8465 8470 8475 8480 8485 8490 8495 8500 8505 8510 8515 8520 8525 8530 8535 8540 8545 8550 8555 8560 8565 8570 8575 8580 8585 8590 8595 8600 8605 8610 8615 8620 8625 8630 8635 8640 8645 8650 8655 8660 8665 8670 8675 8680 8685 8690 8695 8700 8705 8710 8715 8720 8725 8730 8735 8740 8745 8750 8755 8760 8765 8770 8775 8780 8785 8790 8795 8800 8805 8810 8815 8820 8825 8830 8835 8840 8845 8850 8855 8860 8865 8870 8875 8880 8885 8890 8895 8900 8905 8910 8915 8920 8925 8930 8935 8940 8945 8950 8955 8960 8965 8970 8975 8980 8985 8990 8995 9000 9005 9010 9015 9020 9025 9030 9035 9040 9045 9050 9055 9060 9065 9070 9075 9080 9085 9090 9095 9100 9105 9110 9115 9120 9125 9130 9135 9140 9145 9150 9155 9160 9165 9170 9175 9180 9185 9190 9195 9200 9205 9210 9215 9220 9225 9230 9235 9240 9245 9250 9255 9260 9265 9270 9275 9280 9285 9290 9295 9300 9305 9310 9315 9320 9325 9330 9335 9340 9345 9350 9355 9360 9365 9370 9375 9380 9385 9390 9395 9400 9405 9410 9415 9420 9425 9430 9435 9440 9445 9450 9455 9460 9465 9470 9475 9480 9485 9490 9495 9500 9505 9510 9515 9520 9525 9530 9535 9540 9545 9550 9555 9560 9565 9570 9575 9580 9585 9590 9595 9600 9605 9610 9615 9620 9625 9630 9635 9640 9645 9650 9655 9660 9665 9670 9675 9680 9685 9690 9695 9700 9705 9710 9715 9720 9725 9730 9735 9740 9745 9750 9755 9760 9765 97

el(s) 214, and other programs) and data while they are being used, including the facility and associated data, an operating system including a kernel, and device drivers; a network connection component(s) 216 for the computer system to communicate with other computer systems and to send and/or receive data, such as via the Internet or another network and its networking hardware, such as switches, routers, repeaters, electrical cables and optical fibers, light emitters and receivers, radio transmitters and receivers, and the like; a persistent storage(s) device 218, such as a hard drive or flash drive for persistently storing programs and data; and computer-readable media drives 220 (e.g., at least one non-transitory computer-readable medium) that are tangible storage means that do not include a transitory, propagating signal, such as a floppy, CD-ROM, or DVD drive, for reading programs and data stored on a computer-readable medium. While computer systems configured as described above are typically used to support the operation of the facility, those skilled in the art will appreciate that the facility may be implemented using devices of various types and configurations, and having various components.

FIG. 3 is a system diagram illustrating an example of a computing environment in which the disclosed system operates in some implementations. In some implementations, environment 300 includes one or more client computing devices 302a-d, examples of which can host the platform-agnostic data pipeline 100 (or other system components). For example, the computing devices 302a-d can comprise distributed entities a-d, respectively. Client computing devices 302 operate in a networked environment using logical connections through network 304 to one or more remote computers, such as a server computing device. In some implementations, client computing devices 302 may correspond to device 200 (FIG. 2).

In some implementations, server computing device 306 is an edge server which receives client requests and coordinates fulfillment of those requests through other servers, such as servers 310a-c. In some implementations, server computing devices 306 and 310 comprise computing systems. Though each server computing device 306 and 310 is displayed logically as a single server, server computing devices can each be a distributed computing environment encompassing multiple computing devices located at the same or at geographically disparate physical locations. In some implementations, each server computing device 310 corresponds to a group of servers.

Client computing devices 302 and server computing devices 306 and 310 can each act as a server or client to other server or client devices. In some implementations, server computing devices (306, 310a-c) connect to a corresponding database (308, 312a-c). As discussed above, each server computing device 310 can correspond to a group of servers, and each of these servers can share a database or can have its own database. Databases 308 and 312 warehouse (e.g., store) information such as, predefined ranges, predefined thresholds, error thresholds, graphical representations, code portions (e.g., predetermined code portions, predetermined code portions corresponding to combinations of nodes and links, policy-controlled code portions), system policies or other policies, templates, JSON templates, JSON objects, computing languages, platform identifiers, data structures, software application identifiers, nodes, links, graphical representations of nodes and links, predetermined data pipeline architectures, default data pipeline architectures, pre-generated data pipelines, data pipelines, transformation components, visual layouts, security-assessment values, computing language identifiers, mathematical formulas

(e.g., weighted average, weighted sum, or other mathematical formulas), graphical elements (e.g., colors, shapes, text, images, multimedia), system protection mechanisms, or other information.

Though databases 308 and 312 are displayed logically as single units, databases 308 and 312 can each be a distributed computing environment encompassing multiple computing devices, can be located within their corresponding server, or can be located at the same or at geographically disparate physical locations.

Network 304 can be a local area network (LAN) or a wide area network (WAN) but can also be other wired or wireless networks. In some implementations, network 304 is the Internet or some other public or private network. Client computing devices 302 are connected to network 304 through a network interface, such as by wired or wireless communication. While the connections between server computing device 306 and server computing device 310 are shown as separate connections, these connections can be any kind of local, wide area, wired, or wireless network, including network 304 or a separate public or private network.

Generating Platform-Agnostic Data Pipelines

FIG. 4 a flow diagram illustrating a process of generating a platform-agnostic data pipeline via a low code transformation layer, in accordance with some implementations of the present technology.

At act 402, process 400 can receive user selections of nodes and links. For example, process 400 can receive, via a graphical user interface (GUI), user selections of (i) nodes and (ii) a set of links linking the nodes. For example, the user selections may indicate a data pipeline architecture of a data pipeline. To enable unskilled or otherwise untrained users to create data pipelines, a user may interact with a GUI to enable the user to select nodes, link nodes together, provide commands, instructions, or other criteria to generate a data pipeline without the need to possess the complex knowledge associated with coding a data pipeline from scratch via source code.

Referring to FIG. 5, FIG. 5 shows an illustrative representation of a Graphical User Interface (GUI) for generating a platform-agnostic data pipeline via a low code transformation layer, in accordance with some implementations of the present technology. For example, GUI 500 can be provided via a website associated with a data pipeline service provider. The data pipeline service provider may enable users to create data pipelines via the GUI and in some implementations, when implemented, may generate executable instructions to host the data pipeline on one or more computing devices (e.g., a user device, a server, cloud-based computing network, or other computing device).

To enable users to create data pipelines, a user can interact with GUI 500. GUI 500 can include one or more user selectable components, such as node buttons 502a-502d (or as collectively referred to as node button(s) 502), command buttons 504a-504b (or as collectively referred to as command buttons 504), or other user selectable components. For example, in a first portion 512a of GUI 500, first portion 512a can display the user selectable components, and second portion 512b of GUI 500 can enable users to create a data pipeline architecture 510. For instance, first node button 502a may indicate a data source node (e.g., a data repository), second node button 502b may indicate a get operation node, third node button 502c may indicate a filter operation node, fourth node button 502d may indicate a join operation node. First command button 504a may indicate a new link button, second command button 504b may indicate a command to enable a user to open a pre-existing pipeline, third

command button **504c** may indicate a save pipeline button (e.g., to save the current data pipeline architecture **510** in second portion **512b** of GUI **500**), and fourth command button **504d** may indicate an implement pipeline button (e.g., to run, host, or generate executable instructions to create the data pipeline architecture **510**). It should be noted that the user selectable components shown in FIG. 5. are exemplarily and that other user selectable components (e.g., nodes, command buttons, etc.) may exist, in accordance with one or more implementations of the present technology.

In some implementations, a user may select the user selectable components to create (or otherwise generate a data pipeline). For example, a user may select user selectable components in a “drag-and-drop” fashion to create a data pipeline architecture. As another example, a user may select user selectable components and the selected user selectable component may be populated into the data pipeline architecture in one or more default positions (e.g., to enable the user to forgo dragging-and-dropping a user selectable component). By creating data pipelines via a GUI, the system generates for display, in a visual layout, the data pipeline architecture **510** of the created data pipeline to enable users to view the architecture of a data pipeline which reduces errors involved with creating data pipelines otherwise missed by conventionally coding data pipelines using source code. In this way, the system provides an easily understood GUI to enable unskilled users to create data pipeline architectures, thereby improving the user experience. Additionally, in this way, the system provides a GUI to enable users to edit or alter a data pipeline architecture without needing to compile or recompile each variation of the data pipeline, thereby reducing the amount of computer processing and computer memory resources otherwise required to edit a data pipeline.

In one use case, a user can create a data pipeline architecture **510** using user selectable components. The user may select one or more of node buttons **502** from first portion **512a** of the GUI **500** and drag them into a position in a second portion **512b** of the GUI **500**. For instance, the user can select node button **502a** and drag an instance of the node button **502a** into the second portion **512b** of the GUI **500** to place first node **506a**. The user may then provide first node **506a** with a label (e.g., Calendar) to indicate a data source (or database/data repository) from which data will be retrieved from. Similarly, the user can select node button **502b** to drag an instance of the node button **502b** into the second portion **512b** of the GUI **500** to place second node **506b**. The user may also provide second node **506b** with an instruction (e.g., get date) to retrieve one or more dates from a given node. The user can also select command button **504a** to create a new link. For example, the user may select command button **504a** which may enable the user to draw, drag, drop, or otherwise create links **508**. In this example, the user may draw a link connecting first node **506a** and second node **506b** together via first link **508a**. As such, the user may repeat such actions to create a data pipeline architecture **510**. For instance, the user may connect the nodes **506** via links **508** (e.g., including fourth node **506d**, such as output node) to create the data pipeline architecture **510**. In this way, the system enables unskilled users to efficiently and effectively create data pipelines through a GUI as opposed to writing source code, compiling, and testing the data pipeline, thereby improving the user experience.

In some implementations, a user selected node can include an instruction to generate a data structure. For example, to enable a user to analyze results associated with

a data pipeline (e.g., retrieved data), the retrieved data may be stored in a data structure to be accessible to the user. As such, a user may select a node button that may be associated with an instruction/command to generate a data structure. 5 For instance, upon selecting a node button, such as first node button **502a**, and placing the selected node into position (e.g., fourth node **506d**, such as a data output node), the user may provide fourth node **506d** with an instruction/command to generate a data structure to store the retrieved data from 10 the data pipeline architecture **510**. In some implementations, the data structure can be a default data structure (e.g., a table, a tree, an array, etc.) or the data structure may be indicated (e.g., provided) by the user to enable the user to indicate which type of data structure (or format) they wish to output 15 the data from the data pipeline architecture **510**. Additionally, in some implementations, the user may further provide an instruction/command indicating a particular database to store the data retrieved by the data pipeline architecture **510**. In this way, the system provides an easy-to-use user interface to enable users of varying knowledge backgrounds to 20 create robust data pipelines, thereby improving the user experience.

At act **404**, process **400** can generate a JSON object associated with the user selections. For example, process **400** can generate, using the user selections, a JSON object associated with the user selections indicating the data pipeline architecture. JSON (e.g., JavaScript Object Notation) is a format for storing and transporting data. The advantage of 25 using JavaScript and generating a JSON object indicating the data pipeline architecture is to enable user-created data pipelines to be generated for any platform, regardless of the computing requirements of the given platform (e.g., the processors, instruction sets, hardware, software, etc.). For instance, a JSON object may be generated (e.g., via a 30 website, webpage, web browser, or graphical user interface) to indicate the data pipeline architecture created by a user. As an example, the JSON object may indicate a template of the user selections (e.g., the nodes, the links linking the 35 nodes, the user provided instructions/commands associated with the nodes, etc.) to enable a platform agnostic data 40 pipeline to be generated.

As an example, upon creating a data pipeline via GUI **500** (FIG. 5), a user may select a command button **504** (e.g., third command button **504c**) to save the data pipeline. Upon 45 saving the data pipeline, process **400** may generate a JSON object of the data pipeline architecture **510** indicating a template of data pipeline architecture **510**. For instance, the JSON object may include one or more key value pairs indicating nodes, instructions associated with the nodes, 50 labels associated with the nodes, commands associated with the nodes, or other information of the data pipeline architecture. Additionally, the JSON object may further include information regarding the links of the data pipeline architecture **510**, to indicate what nodes are connected together. 55 As a JSON object uses a small amount of memory (e.g., 1 kb, 2 kb, 3 kb, etc.), the system reduces the amount of computing memory required to save/store data pipeline architectures as opposed to existing systems that require large source code files to implement/create data pipelines. 60 Additionally, by using JSON objects indicating data pipeline architectures, data pipelines may be shared amongst users quickly and efficiently over wired/wireless computing networks (e.g., the Internet), thereby reducing the amount of network traffic and network bandwidth traditionally utilized to share fully implemented data pipelines.

At act **406**, process **400** can receive a user selection to implement a data pipeline on a remote execution platform.

11

For example, process **400** can receive a user selection to implement the data pipeline, where the data pipeline is to be hosted on a remote execution platform. For instance, a user may select a command button **504** (e.g., fourth command button **504d**) to implement the data pipeline. The data pipeline may be implemented on a remote execution platform, such as a remote computing platform (e.g., remote server, remote computer, remote network, remote cloud computing system, etc.). For example, to reduce the amount of computer processing resources conventionally required to implement a data pipeline, a user can implement a data pipeline to be hosted on a remote server. The remote server may be part of a data pipeline service provider's computing network that may be configured to host data pipelines. Additionally, as a user may be creating a data pipeline architecture on a mobile device (e.g., smartphone), via a GUI (e.g., GUI **500**), the user may indicate a remote execution platform to host (or otherwise implement/run) the data pipeline to reduce the amount of computational resources used at the mobile device. For instance, as a mobile device may not be configured with a robust processor or large memory, the user may implement the data pipeline on a remote execution platform. In some implementations, however, the data pipeline may be implemented on a non-remote execution platform (e.g., implemented locally on a device the user is currently using). For example, where the user is using a desktop computer, the data pipeline may be implemented on the desktop computer. In this way, the system enables various platforms to implement or host the data pipeline (e.g., where computer processor and memory resources may be sufficient or lacking), to allow a multitude of options to host the data pipeline, irrespective of computational requirements of a device the user is using to create the data pipeline architecture.

At act **408**, process **400** can determine a set of code portions. For example, in response to receiving the user selection to implement the data pipeline, process **400** can determine, using the JSON object, a set of code portions. Each code portion of the set of code portions can correspond to a respective combination of at least (i) a first and a second node of the user selections and (ii) a link linking the first and second nodes. For instance, process **400** may detect a user selection to implement the data pipeline, and may cause, during a runtime routine, the identification of a platform identifier associated with the remote execution platform, the determination of the set of code portions (e.g., predetermined policy-controlled code portions), and the generation of executable instructions to implement the data pipeline. For example, by performing such steps during a runtime routine, the system may reduce the amount of computing processing and memory resources conventionally required to generate a data pipeline as runtime routines are performed at runtime (e.g., and do not require specialized compilers for a given computing language). That is, runtime routines do not rely on a compiler and may be generated quickly and efficiently as opposed to conventional computer programs requiring source code to be compiled and then executed on a specific platform, affecting the scalability and shareability of data pipelines (or other computer programs).

To generate a platform-agnostic data pipeline (e.g., to be hosted on a given platform), process **400** can identify which execution platform the data pipeline is to be hosted on. For instance, as each execution platform (e.g., computing device, computing system, server, servers, etc.) may each be associated with their own computational requirements (e.g., processors, software, hardware, memory requirements, security requirements, etc.), process **400** may identify which

12

execution platform the data pipeline is to be executed on. By doing so, process **400** may determine the correct set of code portions to use when generating executable instructions for implementing the data pipeline.

As discussed above, the code portions can be predetermined policy-controlled code portions. For example, a service provider may use a policy for controlling which data repositories, sources, and actions (e.g., operations such as filtering, selecting, joining, etc.) on the data may be implemented. The policy may indicate security requirements or permissions that are associated with implementing the data pipeline. For example, as data pipelines can be used to obtain sensitive user data, the policy may provide requirements and other rules pertaining to use of the user data. As such, each code portion may be a pre-vetted code portion that conforms to one or more regulations, rules, or other policies as defined by the service provider and/or industry. Moreover, each code portion may be associated with a security assessment value indicating a level of security associated with the remote execution platform. For example, as described above, not only may the policy indicate security requirements associated with implementing the data pipeline, but the platform at which the data pipeline is to be executed on may also have its own security vulnerabilities. As such, when implementing a data pipeline, it is advantageous to be aware of how a code portion (e.g., the source code) for implementing a data pipeline may interact with a given platform. As such, each code portion may be associated with a predetermined security assessment value (e.g., score, value, quantitative value, qualitative value) indicating how secure the given code portion is with respect to a given execution platform. In some implementations, process **400** may generate for display the security assessment value via the GUI to notify the user of how secure the data pipeline to be implemented on the execution platform is. In this way, the user may be provided with security information to aid their decision as to implement the data pipeline on the given execution platform. Furthermore, in this way, the system may pre-vet access or other operations performed on data to reduce data breaches of sensitive user data, thereby improving the security of data obtained via data pipelines. Moreover, in this way, the system reduces the amount of computer processing and memory resources conventionally utilized during a manual review of ethical data harvesting by using the predetermined, vetted, policy-controlled code portions.

Additionally, each code portion of the set of code portions can correspond (or otherwise be associated with) a respective combination of a link of a set of links and nodes connected via the link. For instance, referring back to FIG. 5, as the JSON template (or JSON object) can indicate the architecture of the data pipeline, the JSON template may not include the actual code used to generate (or implement) the data pipeline. For example, as the JSON template may include node-related, link-related, or operation-related information of the data pipeline architecture, such as node identifiers, link identifiers, user selectable commands/instructions, data types, data labels, or other information, the JSON template may represent placeholder values for source code to be implemented. That is, to reduce the amount of computer memory conventionally utilized to edit, share, and implement data pipelines, the JSON template may be used as a template for which source code may be combined with to generate the data pipeline itself. For example, since the JSON template uses a minuscule amount of computer memory, the JSON template may be shared amongst other users (e.g., over one or more computing networks) while reducing the amount of bandwidth and network traffic

conventionally required to share large data pipeline-related files. As such, the JSON template may be easily sharable, however, may not include executable instructions for executing the implementation of the data pipeline itself.

Each code portion of the set of code portions, however, can correspond to a respective combination of the nodes and links of data pipeline architecture **510**. For instance, the set of code portions may be predetermined code portions that include placeholder values (e.g., data types, labels, operations, user provided instructions/commands, etc.) for each node and link combination of the data pipeline architecture **510**. For example, the JSON template may include an indication for first node **506a** to be linked to second node **506b**. For each of the nodes indicated in the JSON template, each node may be associated with a label, a data type, or one or more user provided/selected commands or instructions. For instance, in the JSON template, the first node **506a** may be associated with a label of “Calendar” (e.g., identifying the data repository to which data is obtained from) and a data type of “data source” (e.g., indicating the node is associated with a data repository or data base). Second node **506b** may be associated with a label of “Date” (e.g., indicating to obtain date(s) from a data source), a data type of “Select” (e.g., indicating the operation of the node is a select operation), and one or more user provided instructions/commands (e.g., criteria). For example, the one or more user provided instructions/commands may be used to instruct the node a date range of the data to obtain, or other dates, in accordance with one or more implementations of the present technology. As yet another example, the JSON template may also include an indication that first node **506a** is linked to second node **506b**.

Process **400** can use the JSON template to determine a code portion that corresponds to the combination of first node **506a** and second node **506b** being linked. For example, process **400** can select a predetermined code portion that links together a first node and a second node. Process **400** can use the data types indicated in the JSON template to identify a code portion that links together nodes of the respective data types. For instance, a predetermined code portion may link together a data type of “data source” and a data type of “select.” Process **400** can parse through the set of predetermined code portions using the data types as indicated in the JSON template to determine a match between the data types of the predetermined code portions. Upon identifying a match, process **400** may select the predetermined code portion. It should be noted that other combinations of corresponding code portions linking nodes together may exist, and such example is exemplary.

To clarify, the JSON template may include information that is related to the data pipeline architecture **510**, however, such information may not be executable within a computing environment. The predetermined code portions, however, include executable code that includes additional implementation details on how given nodes are linked (or otherwise connected) to other nodes, or additional input parameters from the JSON template (e.g., labels, criteria, user selectable commands, etc.). For instance, to enable users to create robust data pipelines who may be unskilled or unknowledgeable of how data pipelines are coded/implemented via source code, the predetermined code portions represent “implementable code” that may be pieced together to execute the created data pipeline architecture **510**. That is, the predetermined code portions may be pieces of source code that may be put together, using labels included in the JSON template, to indicate which data sources data is being retrieved from, what user provided commands are associated

with given nodes, what operations to perform on the data being obtained from a given source/node, and how the nodes are linked together. Furthermore, as each code portion may be predetermined, such code portions are tested and validated for functionality ensuring that when a data pipeline is created using the predetermined code portions, the data pipeline will be executed without error. In this way, the system optimizes data pipeline generation by using predetermined and pre-vetted (e.g., for errors) code portions as opposed to existing systems testing data pipeline implementations, thereby reducing the amount of computer processing and computer memory resources required to implement a data pipeline.

At act **410**, process **400** can generate executable instructions associated with the data pipeline. For example, process **400** can generate executable instructions associated with the data pipeline architecture by using a transformation component configured to process the set of code portions with respect to the remote execution platform using the JSON object (e.g., template). As discussed above, process **400** may use the set of code portions to generate executable instructions for implementing the data pipeline. The executable instructions may be customized for the remote execution platform. For example, as the remote execution platform may be associated with its own requirements, the executable instructions may be customized to run (e.g., execute, host, or otherwise implement) the data pipeline.

In some implementations, process **400** can use a transformation component to generate the executable instructions associated with the data pipeline architecture. The transformation component may transform the JSON object and the associated code portions (e.g., the code portions corresponding to the JSON template) into executable instructions to execute on the remote execution platform. The transformation component may be part of a computing device (e.g., client computing device **302** (FIG. 3)) and may be hosted or installed on the computing device via a web-browsing application or other software application (e.g., application **212a** of device **200** (FIG. 2)). In other implementations, the transformation component may be hosted on the web server providing the GUI, in accordance with some implementations of the present technology. For example, the transformation component may be a low code transformation component (e.g., as implemented as part of a software application) or may be a low code transformation layer (e.g., as implemented via a website or web server associated with the system). The benefit of using a low code transformation component is realized by the mechanism it operates. For example, the low code transformation component (e.g., transformation component), as discussed above, uses “low code” solutions to implement robust computing programs, such as data pipelines. Using low code platforms enables users that do not possess the specialized knowledge to implement robust software applications that were otherwise unattainable. By using the predetermined code portions, the low code transformation layer can generate and implement data pipelines (and other software applications) quickly and efficiently as opposed to existing systems.

The transformation component may be configured to receive, as input, the JSON template, the predetermined code portions, or the platform identifier and output executable instructions customized for the execution platform (e.g., the remote execution platform). Additionally, the transformation component may process the inputs (e.g., of one computing language, of multiple computing languages, such as JavaScript, Python, C, C++, Java, Ruby, etc.) and generate the executable instructions via runtime technolo-

gies such as Spark, Pyspark, Beam, Storm, or other runtime technologies. In this way, the system may generate platform-agnostic data pipelines as the executable instructions for implementing the data pipelines are generated at runtime using predetermined code portions, the JSON template, and the platform identifier, thereby reducing the amount of computer processing and memory resources conventionally required to implement a data pipeline. Moreover, by generating such executable instructions during a runtime routine for any given platform, the system ensures widespread scalability and shareability of such pipelines as edits, updates, or changes to such data pipelines may be pushed and generated at runtime, thereby further reducing the amount of network traffic and computer processing and memory resources.

The transformation component can use the JSON template to piece together the predetermined code portions and generate the executable instructions in a computing language readable by the remote execution platform. For example, as discussed above, the JSON template may include information that is associated with the user selections indicating the data pipeline architecture (e.g., nodes, links, labels, user provided instructions/commands, data types, etc.). The transformation component may use such information and the predetermined code portions corresponding to the user selections to generate the executable instructions during a runtime routine. The advantage of the transformation component as opposed to existing systems is that the transformation component uses a generic computing language (e.g., which the JSON template and the predetermined code portions are written in) and transforms the generic computing language into a computing language understood by the platform to which the data pipeline is to be implemented (e.g., hosted) on. That is, as opposed to requiring a specialized compiler to compile source code to be executable on a given platform, the transformation component uses the generic language of the JSON template and the predetermined code portions to generate executable instructions at runtime, in a language understood by the remote execution platform. In this way, the system enables data pipelines to be implemented on any platform, regardless of the technical requirements of the execution platform.

In some implementations, a transformation component may be selected from a set of transformation components. For example, in some implementations, transformation components may be specific to one or more computing languages. As such process 400 may determine a first computing language associated with the remote execution platform. For instance, process 400 may query a database including platform identifiers (e.g., a value, alphanumeric value, name, or other identifier) and associated computing languages. Each platform identifier may correspond to a given computing language (e.g., JavaScript, Python, C, C++, Ruby, Java, etc.). Process 400 may determine a match between the platform identifier of the remote execution platform (e.g., to host the data pipeline) and a platform identifier of the database to identify the computing language associated with the remote execution platform. Process 400 may then determine a second computing language associated with the set of code portions. For example, the set of predetermined code portions may be written in another computing language which may differ from that of the remote execution platform. Therefore, to generate the executable instructions associated with the data pipeline architecture, process 400 may select a transformation component from a set of transformation components that is associated with (i) the first computing language and (ii) the second computing language. The set of

transformation components may be stored in a database (e.g., database 308), and process 400 may query the database using the first computing language and the second computing language to select a transformation component to be used for generating the executable instructions. In some implementations, the selected transformation component may be configured to transform each code portion of the set of code portions associated with the second computing language into the first computing language (e.g., of the remote execution platform). Process 400 may then generate, using the selected transformation component, the executable instructions associated with the data pipeline by providing the selected transformation component with the set of code portions and the JSON template. In this way, the system may generate platform-agnostic data pipelines by transforming code of one computing language into another.

At act 412, process 400 can provide executable instructions to the remote execution platform. For example, process 400 can provide the executable instructions to the remote execution platform to host the data pipeline. To enable the data pipeline to be hosted on (e.g., executed on, ran on, process data on, etc.), process 400 may automatically transmit (e.g., via the Internet, one or more computing networks, etc.) the executable instructions to the remote execution platform. Additionally, process 400 can automatically execute (e.g., run the executable instructions) on the remote execution platform. In this way, the platform-agnostic data pipeline is automatically implemented on the remote execution platform, thereby reducing data pipeline implementation time delays and enabling users to obtain and analyze data faster than that of conventional systems.

In some implementations, process 400 can provide visual indications related to the data pipeline via a GUI. For example, referring back to FIG. 5, process 400 can provide visual indications of how many times a node is being used with respect to other nodes of other data pipelines. For instance, to provide users (e.g., data analysts, business professionals, etc.) with information regarding lineage tracking of data sources, databases, and data repositories being utilized, process 400 may generate visual indications of an amount of use regarding a given node. For instance, visual indicator 514 may indicate that the Calendar data source has been used twice with respect to other pre-generated data pipelines. That is, such visual indications may provide useful information to users building a data pipeline as to which data repositories are being utilized the most, and which data repositories should be focused on when analyzing data. Not only can such visual indications may provide useful information directed to which data repositories are utilized, but also how certain nodes, filters, or operations are being used in conjunction with the data pipelines and which nodes are in common with other data pipelines implemented via the system.

To accomplish this, process 400 may determine, from a set of pre-generated data pipelines, a set of common nodes. The set of common nodes may indicate nodes that are common to (i) nodes of the set of pre-generated data pipelines and (ii) the data pipeline (e.g., being created/implemented by the user). For example, the system may store JSON objects (e.g., templates) of the pre-generated data pipelines in a database associated with the system (e.g., database 308 (FIG. 3)). In this way, as opposed to storing the fully implemented source code of the pre-generated data pipelines, the system reduces the amount of computer memory required to store the pre-generated data pipelines. Process 400 may determine the set of common nodes by comparing each node of the data pipeline (e.g., the nodes of

data pipeline architecture 510) to each node of the pre-generated data pipelines. Upon determining the set of common nodes, process 400 may determine an amount of common nodes respective to a given common node. For example, for a given node (e.g., first node 506a), process 400 may extract the label associated with the node (e.g., Calendar) and determine how many times the calendar node is used with respect to the pre-generated data pipelines. Process 400 may then generate for display, at the GUI (e.g., GUI 500), a visual indication indicating the amount of common nodes for the given common node. For example, the visual indication may be a numerical value, a color-coded shape, a geometrical object, or other visual indicator indicating the amount of times a given node is used with respect to the pre-generated data pipelines. In this way, the system provides users with lineage tracking information with respect to the usage of data repositories of data pipelines, thereby improving the user experience.

CONCLUSION

Unless the context clearly requires otherwise, throughout the description and the claims, the words "comprise," "comprising," and the like are to be construed in an inclusive sense, as opposed to an exclusive or exhaustive sense; that is to say, in the sense of "including, but not limited to." As used herein, the terms "connected," "coupled," or any variant thereof means any connection or coupling, either direct or indirect, between two or more elements; the coupling or connection between the elements can be physical, logical, or a combination thereof. Additionally, the words "herein," "above," "below," and words of similar import, when used in this application, refer to this application as a whole and not to any particular portions of this application. Where the context permits, words in the above Detailed Description using the singular or plural number may also include the plural or singular number respectively. The word "or," in reference to a list of two or more items, covers all of the following interpretations of the word: any of the items in the list, all of the items in the list, and any combination of the items in the list.

The above Detailed Description of examples of the technology is not intended to be exhaustive or to limit the technology to the precise form disclosed above. While specific examples for the technology are described above for illustrative purposes, various equivalent modifications are possible within the scope of the technology, as those skilled in the relevant art will recognize. For example, while processes or blocks are presented in a given order, alternative implementations can perform routines having steps, or employ systems having blocks, in a different order, and some processes or blocks can be deleted, moved, added, subdivided, combined, and/or modified to provide alternative or sub-combinations. Each of these processes or blocks can be implemented in a variety of different ways. Also, while processes or blocks are at times shown as being performed in series, these processes or blocks can instead be performed or implemented in parallel or can be performed at different times. Further, any specific numbers noted herein are only examples: alternative implementations can employ differing values or ranges.

The teachings of the technology provided herein can be applied to other systems, not necessarily the system described above. The elements and acts of the various examples described above can be combined to provide further implementations of the technology. Some alternative implementations of the technology may include not only

additional elements to those implementations noted above, but also may include fewer elements.

These and other changes can be made to the technology in light of the above Detailed Description. While the above description describes certain examples of the technology, and describes the best mode contemplated, no matter how detailed the above appears in text, the technology can be practiced in many ways. Details of the system may vary considerably in its specific implementation, while still being encompassed by the technology disclosed herein. As noted above, specific terminology used when describing certain features or aspects of the technology should not be taken to imply that the terminology is being redefined herein to be restricted to any specific characteristics, features, or aspects of the technology with which that terminology is associated. In general, the terms used in the following claims should not be construed to limit the technology to the specific examples disclosed in the specification, unless the above Detailed Description section explicitly defines such terms. Accordingly, the actual scope of the technology encompasses not only the disclosed examples, but also all equivalent ways of practicing or implementing the technology under the claims.

To reduce the number of claims, certain aspects of the technology are presented below in certain claim forms, but the applicant contemplates the various aspects of the technology in any number of claim forms. For example, while only one aspect of the technology is recited as a computer-readable medium claim, other aspects may likewise be embodied as a computer-readable medium claim, or in other forms, such as being embodied in a means-plus-function claim. Any claims intended to be treated under 35 U.S.C. § 112(f) will begin with the words "means for," but use of the term "for" in any other context is not intended to invoke treatment under 35 U.S.C. § 112(f). Accordingly, the applicant reserves the right to pursue additional claims after filing this application to pursue such additional claim forms, in either this application or in a continuing application.

We claim:

1. A system for generating a platform-agnostic data pipeline via a low code transformation layer, the system comprising:
 - at least one processor; and
 - at least one memory coupled to the at least one processor and storing instructions
 - that, when executed by the at least one processor, perform operations comprising:
 - receiving, via a graphical user interface (GUI) within a web-browsing application, user selections of: (a) nodes comprising: (i) a first node indicating a data repository, (ii) a second node indicating a data filter, (iii) a third node indicating an output, and (b) a set of links linking two or more of the nodes, the user selections indicating a data pipeline architecture of a data pipeline;
 - generating, based on the user selections, a JSON template related to the user selections indicating the data pipeline architecture;
 - receiving a user selection to execute the data pipeline, wherein the data pipeline is to be hosted on a remote execution platform;
 - in response to receiving the user selection to execute the data pipeline, performing during a runtime routine:
 - identifying a platform identifier associated with the remote execution platform;
 - determining, based on the JSON template and the identified platform identifier, a set of predeter-

19

mined policy-controlled code portions, wherein each predetermined policy-controlled code portion of the set of predetermined policy-controlled code portions corresponds to a respective combination of at least: (i) a link among the set of links, and (ii) nodes connected via the link; 5 generating, via a transformation component, executable instructions to execute one or more commands associated with the data pipeline architecture based on: (i) the JSON template, (ii) the set of predetermined policy-controlled code portions, and (iii) the platform identifier, wherein the transformation component is configured to convert the JSON template into the executable instructions customized for the remote execution platform; and automatically providing the generated executable instructions to the remote execution platform to host the data pipeline.

2. The system of claim 1, wherein the received user 20 selections of the nodes and the set of links linking the nodes are generated for display in the GUI in a visual layout, wherein the visual layout visually depicts the data pipeline architecture of the data pipeline.

3. The system of claim 1, wherein the instructions when 25 executed by the at least one processor further perform operations comprising:

determining, from a set of pre-generated data pipelines, a set of common nodes, wherein the set of common nodes indicate nodes that are common to (i) nodes of the set of pre-generated data pipelines and (ii) the data pipeline;

determining, for each common node of the set of common nodes, an amount of common nodes respective to a given common node; and

generating for display, at the GUI, a visual indication of the amount of common nodes respective to the given common node.

4. The system of claim 1, wherein the third node includes an instruction to generate a data structure associated with 40 one or more of the user selected nodes.

5. The system of claim 1, wherein the transformation component is part of the web-browsing application.

6. The system of claim 1, wherein each predetermined policy-controlled code portion of the set of predetermined policy-controlled code portions are associated with a security assessment value, wherein the security assessment value indicates a level of computing security associated with the remote execution platform.

7. A method for generating a platform-agnostic data 50 pipeline via a low code transformation layer, the method comprising:

receiving, via a graphical user interface (GUI), user selections of (i) nodes and (ii) a set of links linking the nodes, the user selections indicating a data pipeline architecture of a data pipeline;

generating, using the user selections, a JSON object associated with the user selections indicating the data pipeline architecture;

receiving a user selection to implement the data pipeline, 60 wherein the data pipeline is to be hosted on a remote execution platform;

in response to receiving the user selection to implement the data pipeline:

determining, using the JSON object, a set of code 65 portions, wherein each code portion of the set of code portions corresponds to a respective combina-

20

tion of at least (i) a first and a second node of the user selections and (ii) a link linking the first and second nodes;

generating executable instructions associated with the data pipeline architecture by using a transformation component configured to process the set of code portions with respect to the remote execution platform using the JSON object; and

providing the executable instructions to the remote execution platform to host the data pipeline.

8. The method of claim 7, wherein the received user 10 selections of the nodes and the set of links linking the nodes are generated for display in the GUI in a visual layout, wherein the visual layout visually depicts the data pipeline architecture of the data pipeline.

9. The method of claim 7, further comprising: determining, from a set of pre-generated data pipelines, a set of common nodes, wherein the set of common nodes indicate nodes that are common to (i) nodes of the set of pre-generated data pipelines and (ii) the data pipeline;

determining, for each common node of the set of common nodes, an amount of common nodes respective to a given common node; and generating for display, at the GUI, a visual indication of the amount of common nodes respective to the given common node.

10. The method of claim 7, wherein at least one node of the user selected nodes includes an instruction to generate a data structure associated with one or more of the user selected nodes.

11. The method of claim 7, wherein the transformation component is part of an application associated with the GUI.

12. The method of claim 7, wherein each code portion of the set of code portions are associated with a security assessment value, wherein the security assessment value indicates a level of computing security associated with the remote execution platform.

13. The method of claim 7, wherein generating the executable instructions associated with the data pipeline architecture by using the transformation component further comprises:

determining a first computing language associated with the remote execution platform using a platform identifier associated with the remote execution platform; determining a second computing language associated with the set of code portions;

selecting a first transformation component associated with (i) the first computing language and (ii) the second computing language from a set of transformation components, wherein the first transformation component is configured to transform each code portion of the set of code portions associated with the second computing language into the first computing language; and

generating, by the first transformation component, the executable instructions associated with the data pipeline architecture using the (i) set of code portions and (ii) the JSON object.

14. One or more non-transitory computer-readable media comprising instructions that, when executed by one or more processors, cause operations comprising:

receiving, via a graphical user interface (GUI), user selections of (i) nodes and (ii) a set of links linking the nodes, the user selections indicating a data pipeline architecture of a data pipeline;

21

generating, using the user selections, a JSON object associated with the user selections indicating the data pipeline architecture;

receiving a user selection to implement the data pipeline, wherein the data pipeline is to be hosted on a remote execution platform;

in response to receiving the user selection to implement the data pipeline:

determining, using the JSON object, a set of code portions, wherein each code portion of the set of code portions corresponds to a respective combination of at least (i) a first and a second node of the user selections and (ii) a link linking the first and second nodes;

generating executable instructions associated with the data pipeline architecture by using a transformation component configured to process the set of code portions with respect to the remote execution platform using the JSON object; and

providing the executable instructions to the remote execution platform to host the data pipeline.

15. The media of claim 14, wherein the received user selections of the nodes and the set of links linking the nodes are generated for display in the GUI in a visual layout, wherein the visual layout visually depicts the data pipeline architecture of the data pipeline.

16. The media of claim 14, wherein the operations further comprise:

determining, from a set of pre-generated data pipelines, a set of common nodes, wherein the set of common nodes indicate nodes that are common to (i) nodes of the set of pre-generated data pipelines and (ii) the data pipeline;

determining, for each common node of the set of common nodes, an amount of common nodes respective to a given common node; and

22

generating for display, at the GUI, a visual indication of the amount of common nodes respective to the given common node.

17. The media of claim 14, wherein at least one node of the user selected nodes includes an instruction to generate a data structure associated with one or more of the user selected nodes.

18. The media of claim 14, wherein the transformation component is part of an application associated with the GUI.

19. The media of claim 14, wherein each code portion of the set of code portions are associated with a security assessment value, wherein the security assessment value indicates a level of computing security associated with the remote execution platform.

20. The media of claim 14, wherein generating the executable instructions associated with the data pipeline architecture by using the transformation component further comprises:

determining a first computing language associated with the remote execution platform using a platform identifier associated with the remote execution platform; determining a second computing language associated with the set of code portions;

selecting a first transformation component associated with (i) the first computing language and (ii) the second computing language from a set of transformation components, wherein the first transformation component is configured to transform each code portion of the set of code portions associated with the second computing language into the first computing language; and generating, by the first transformation component, the executable instructions associated with the data pipeline architecture using the (i) set of code portions and (ii) the JSON object.

* * * * *