US 20210020041A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2021/0020041 A1**

Yang et al. (43) **Pub. Date: Jan. 21, 2021**

(54) **PRIVACY-PRESERVING DISTRIBUTED VISUAL DATA PROCESSING**

(71) Applicant: **Intel Corporation**, Santa Clara, CA (US)

(72) Inventors: **Shao-Wen Yang**, San Jose, CA (US); **Yen-Kuang Chen**, Palo Alto, CA (US); **Addicam V. Sanjay**, Gilbert, AZ (US)

(73) Assignee: **Intel Corporation**, Santa Clara, CA (US)

(21) Appl. No.: **16/835,193**

(22) Filed: **Mar. 30, 2020**

Related U.S. Application Data

(63) Continuation of application No. 15/859,324, filed on Dec. 29, 2017, now Pat. No. 10,607,484.

(60) Provisional application No. 62/611,536, filed on Dec. 28, 2017.

Publication Classification

(51) **Int. Cl.**

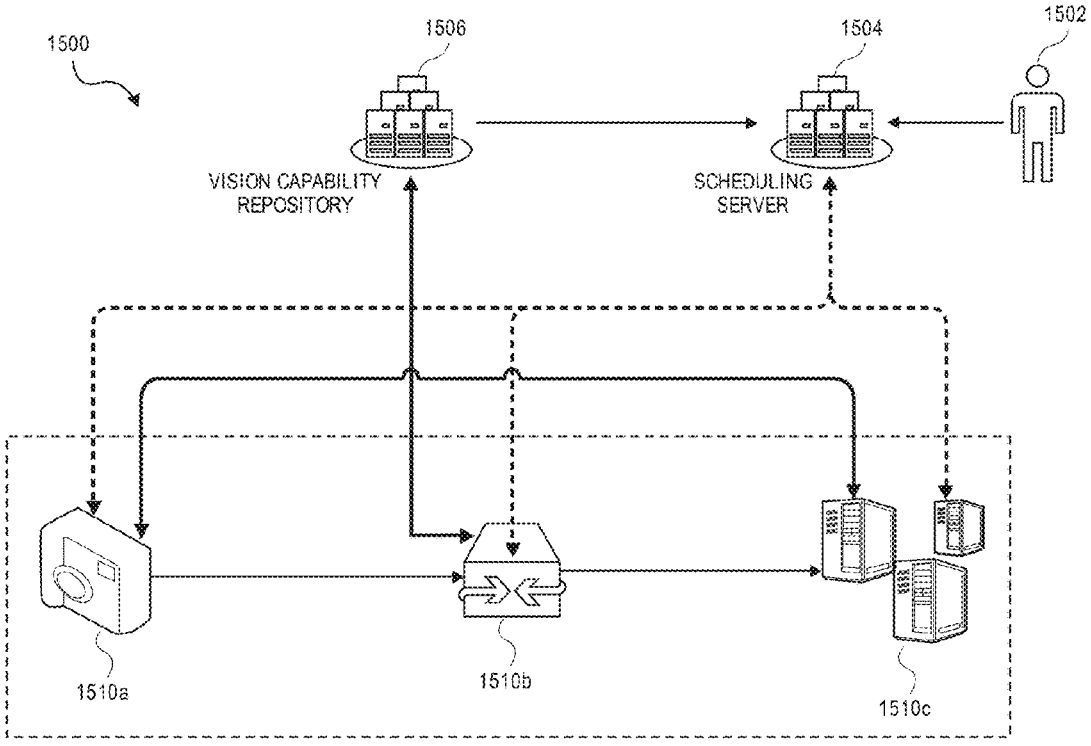
G08G 1/09	(2006.01)
G06F 9/50	(2006.01)
G06F 21/60	(2006.01)
G06K 9/00	(2006.01)
G06F 9/48	(2006.01)
G06F 21/62	(2006.01)
G06K 9/62	(2006.01)
G06K 9/46	(2006.01)
G06Q 50/26	(2006.01)
G11B 27/031	(2006.01)
H04N 7/18	(2006.01)

(52) **U.S. Cl.**

CPC ***G08G 1/091*** (2013.01); *G08G 1/087* (2013.01); ***G06F 21/604*** (2013.01); ***G06K 9/00771*** (2013.01); ***G06F 9/4881*** (2013.01); ***G06F 21/6245*** (2013.01); ***G06K 9/6271*** (2013.01); ***G06K 9/00369*** (2013.01); ***G06K 9/4604*** (2013.01); ***G06K 9/6268*** (2013.01); ***G06Q 50/26*** (2013.01); ***G11B 27/031*** (2013.01); ***H04N 7/181*** (2013.01); *G06F 2209/506* (2013.01); ***G06F 9/505*** (2013.01)

(57) **ABSTRACT**

In one embodiment, an apparatus comprises a processor to: identify a workload comprising a plurality of tasks; generate a workload graph based on the workload, wherein the workload graph comprises information associated with the plurality of tasks; identify a device connectivity graph, wherein the device connectivity graph comprises device connectivity information associated with a plurality of processing devices; identify a privacy policy associated with the workload; identify privacy level information associated with the plurality of processing devices; identify a privacy constraint based on the privacy policy and the privacy level information; and determine a workload schedule, wherein the workload schedule comprises a mapping of the workload onto the plurality of processing devices, and wherein the workload schedule is determined based on the privacy constraint, the workload graph, and the device connectivity graph. The apparatus further comprises a communication interface to send the workload schedule to the plurality of processing devices.



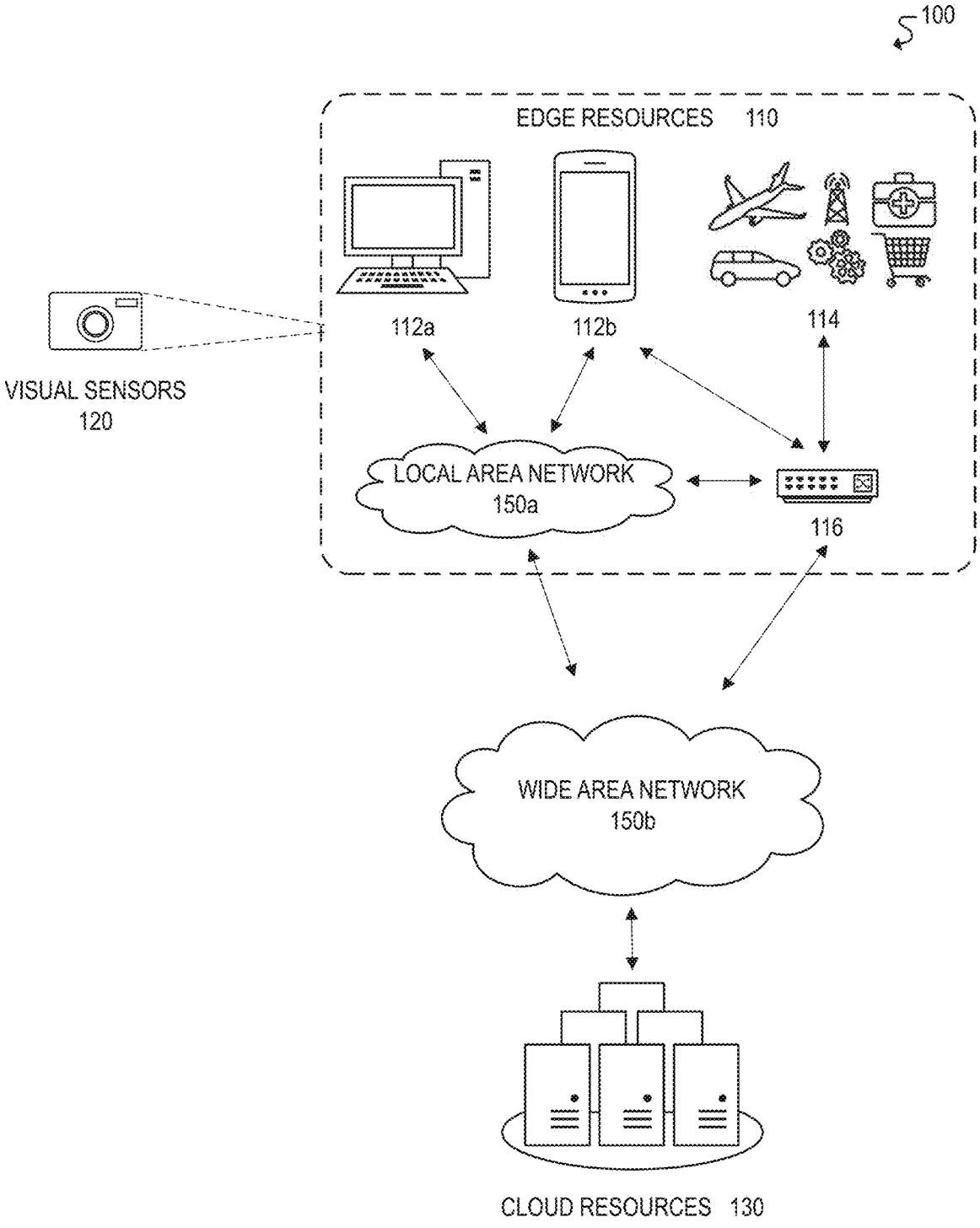

100
EDGE RESOURCES 110
112a
112b
114
VISUAL SENSORS
120
LOCAL AREA NETWORK
150a
116
WIDE AREA NETWORK
150b
CLOUD RESOURCES 130

FIG. 1

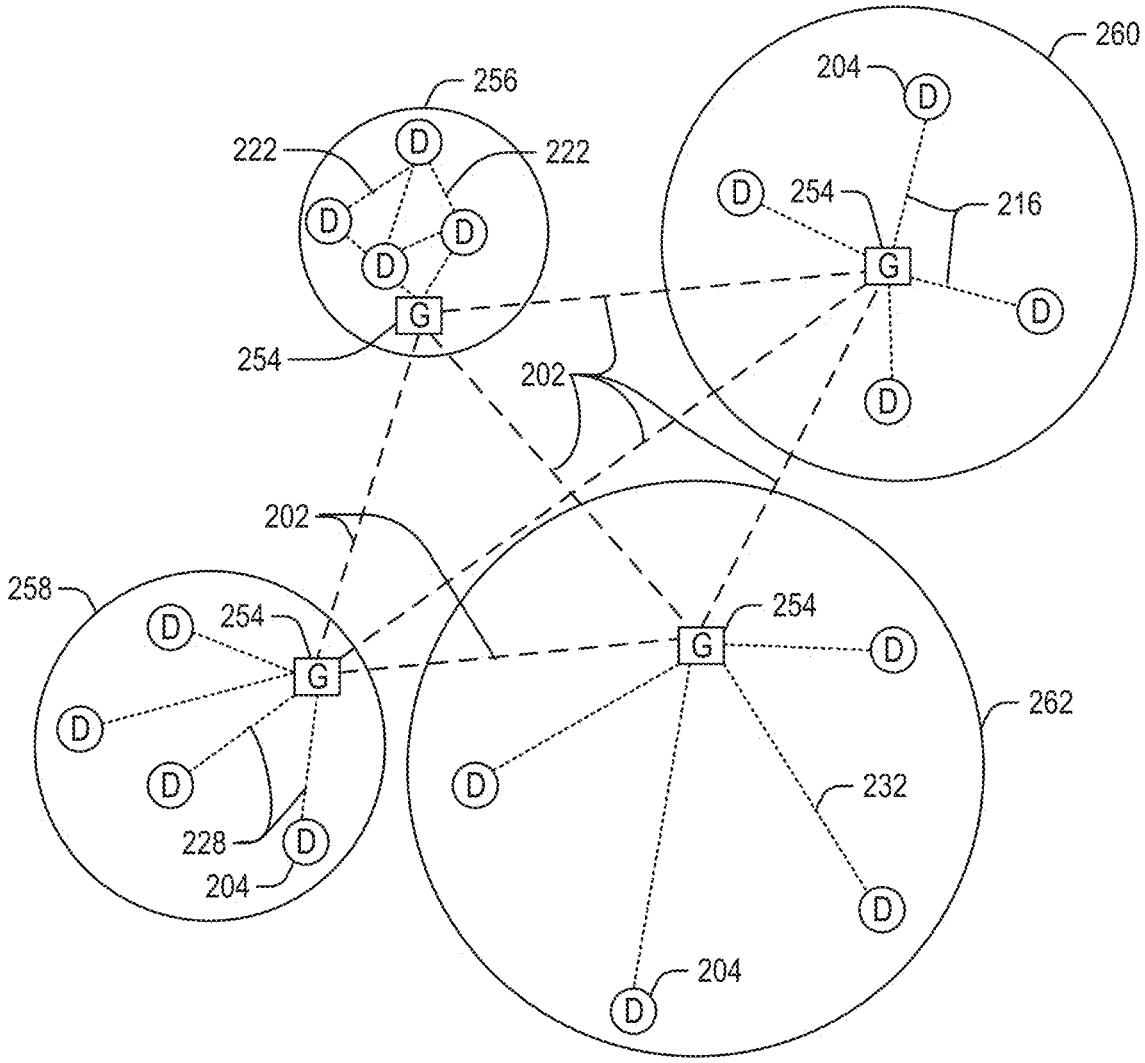

260
204
D
256
222
222
D
D
D
D
D
254
216
G
D
254
202
D
202
258
D
254
G
254
G
D
262
D
D
D
232
228
D
204
D
204
D

FIG. 2

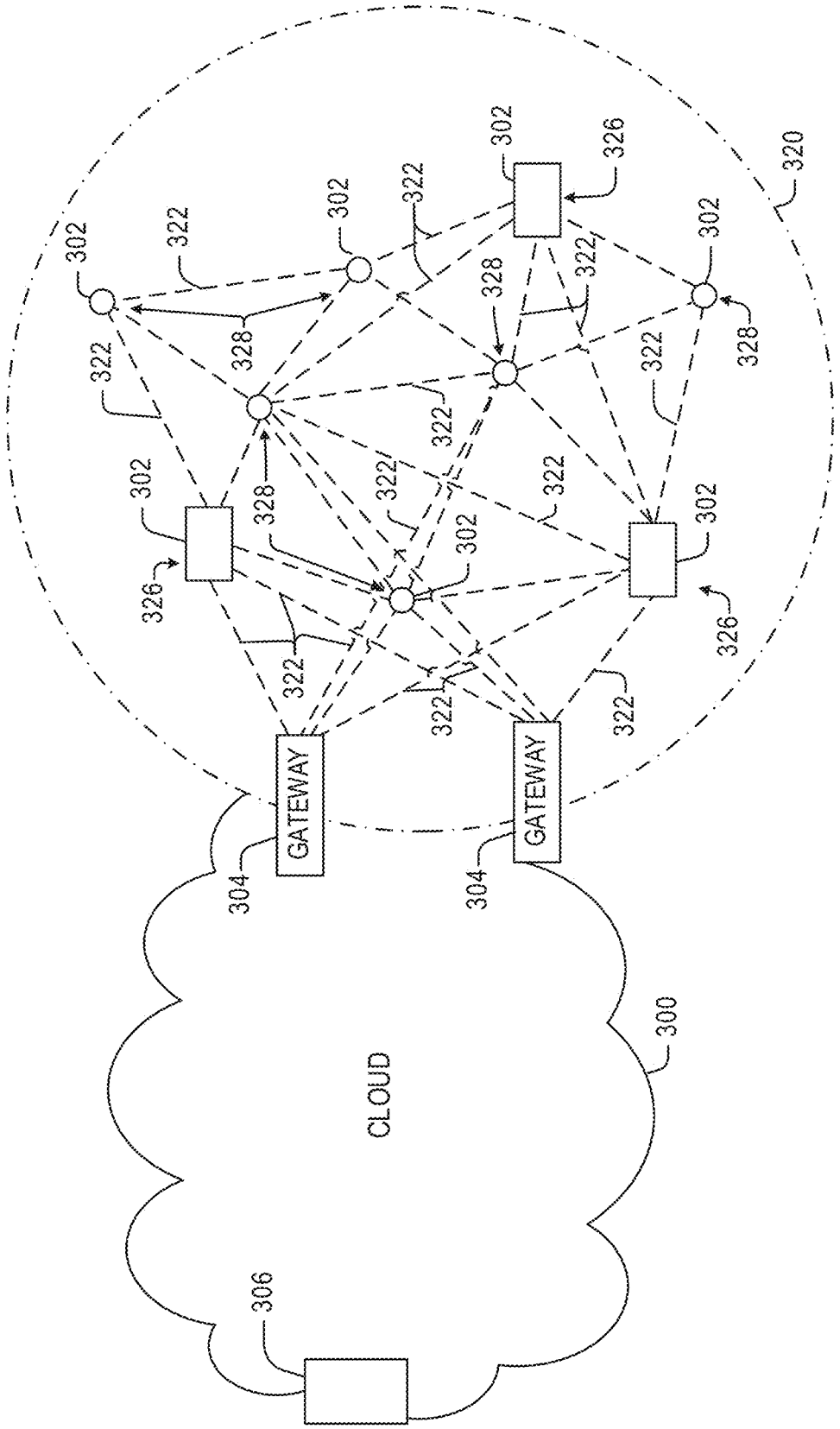


FIG. 3

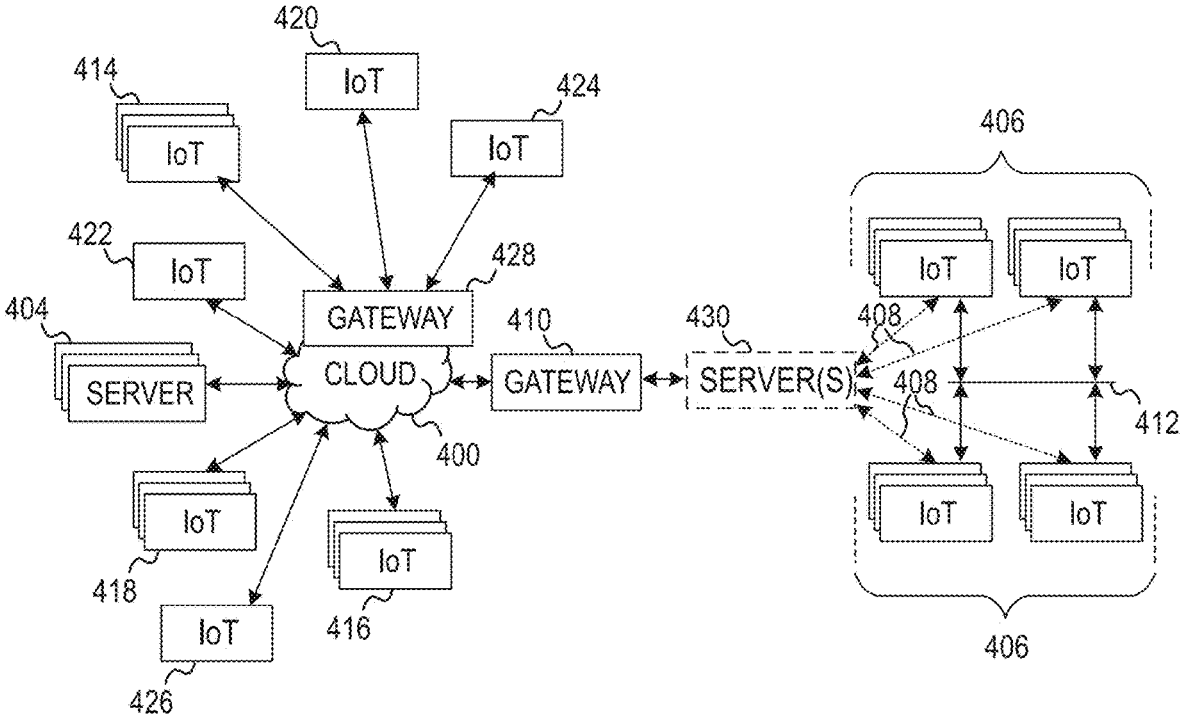

420
IoT
414
IoT
424
IoT
406
422
IoT
428
IoT
IoT
404
GATEWAY
410
430
408
SERVER
CLOUD
GATEWAY
SERVER(S)
408
412
400
IoT
IoT
IoT
418
IoT
416
406
426

FIG. 4

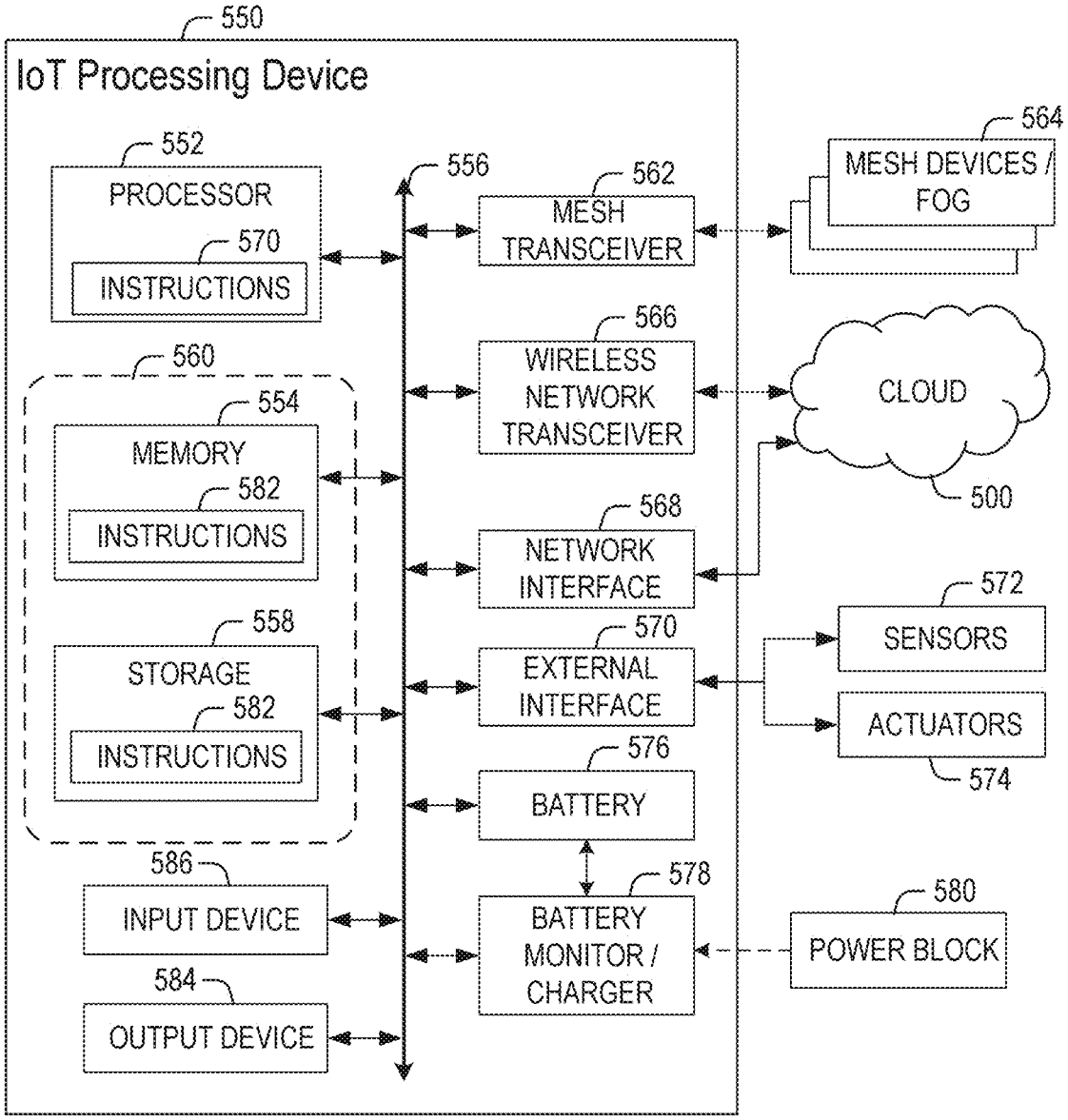

550
IoT Processing Device
552
PROCESSOR
570
INSTRUCTIONS
560
554
MEMORY
582
INSTRUCTIONS
558
STORAGE
582
INSTRUCTIONS
586
INPUT DEVICE
584
OUTPUT DEVICE
556
562
MESH TRANSCEIVER
566
WIRELESS NETWORK TRANSCEIVER
568
NETWORK INTERFACE
570
EXTERNAL INTERFACE
576
BATTERY
578
BATTERY MONITOR / CHARGER
564
MESH DEVICES / FOG
CLOUD
500
572
SENSORS
ACTUATORS
574
580
POWER BLOCK

FIG. 5

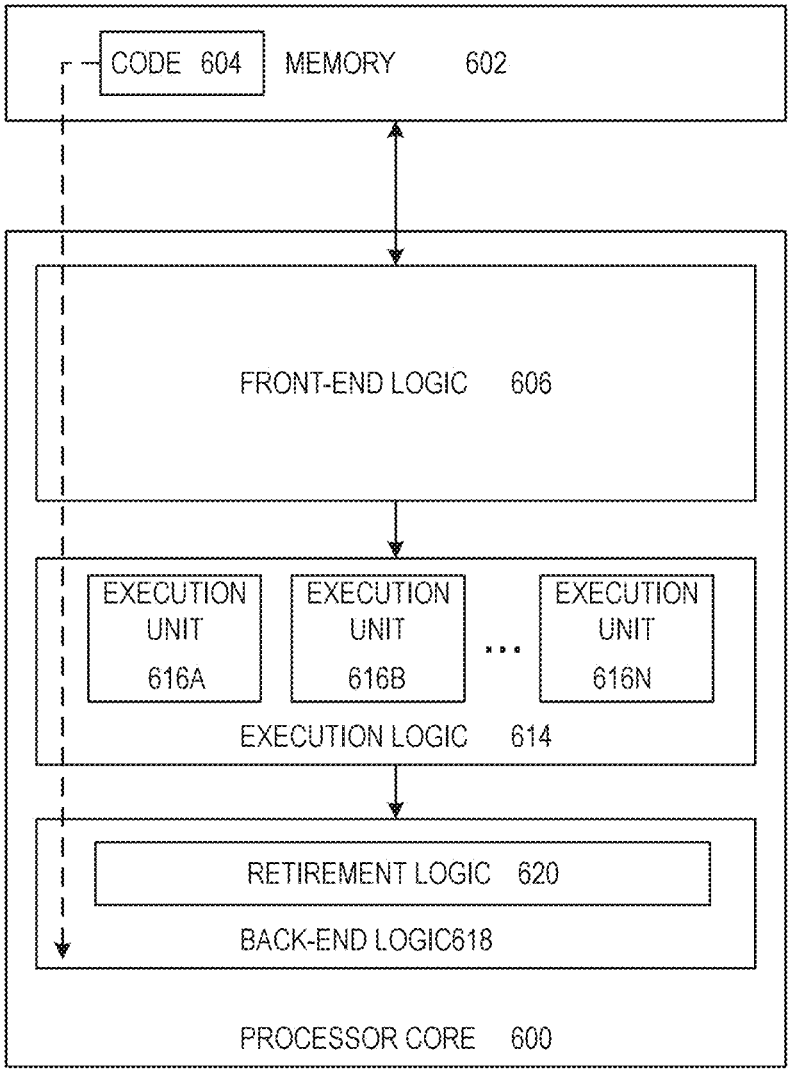

CODE 604
MEMORY 602
FRONT-END LOGIC 606
EXECUTION UNIT 616A
EXECUTION UNIT 616B
EXECUTION UNIT 616N
EXECUTION LOGIC 614
RETIREMENT LOGIC 620
BACK-END LOGIC618
PROCESSOR CORE 600

FIG. 6

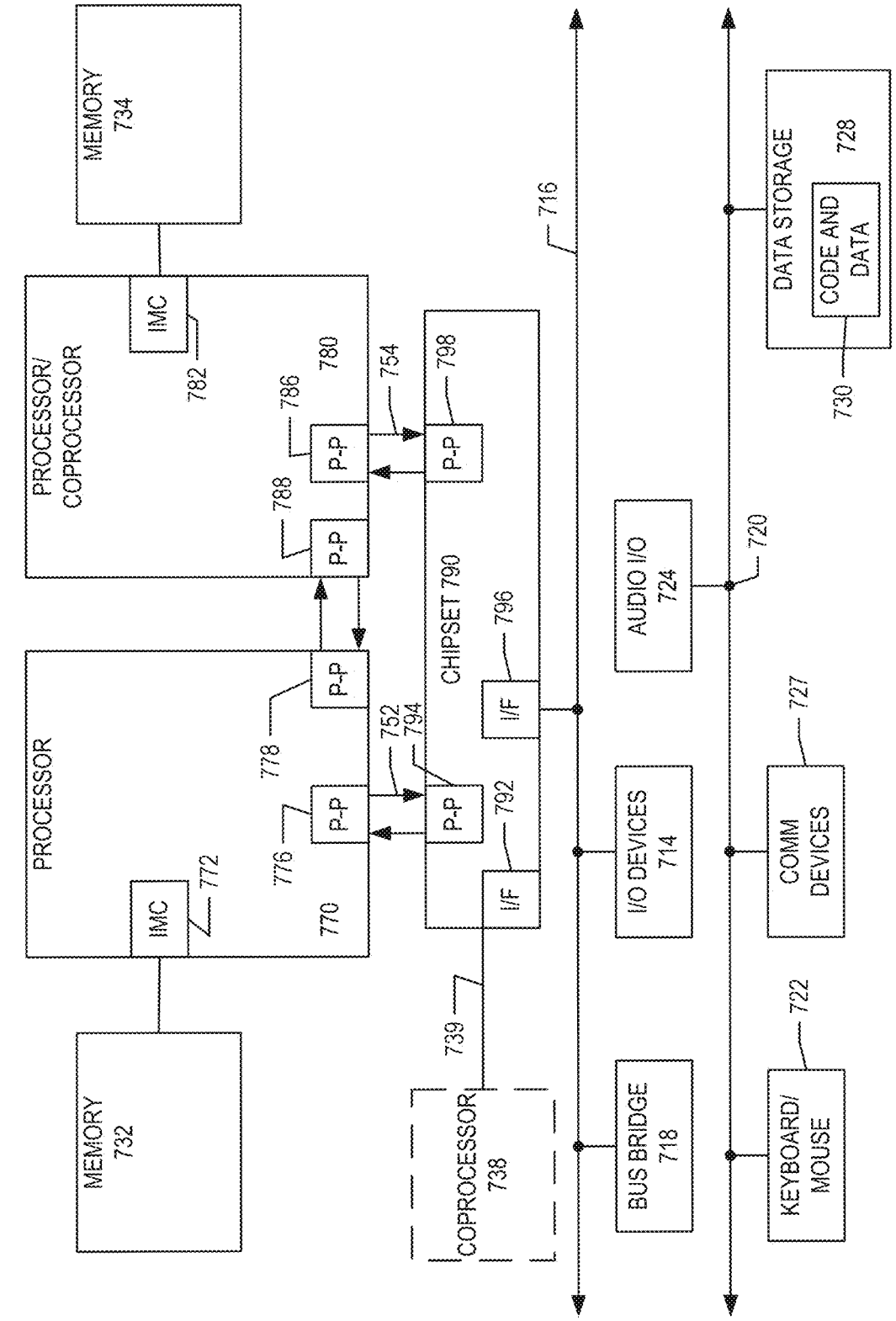

700
MEMORY 732
MEMORY 734
PROCESSOR
PROCESSOR/ COPROCESSOR
IMC
772
782
770
780
P-P
776
778
786
788
752
754
794
792
CHIPSET 790
I/F
796
798
739
COPROCESSOR 738
716
BUS BRIDGE 718
I/O DEVICES 714
AUDIO I/O 724
720
COMM DEVICES
727
KEYBOARD/ MOUSE
722
DATA STORAGE
CODE AND DATA
728
730

FIG. 7

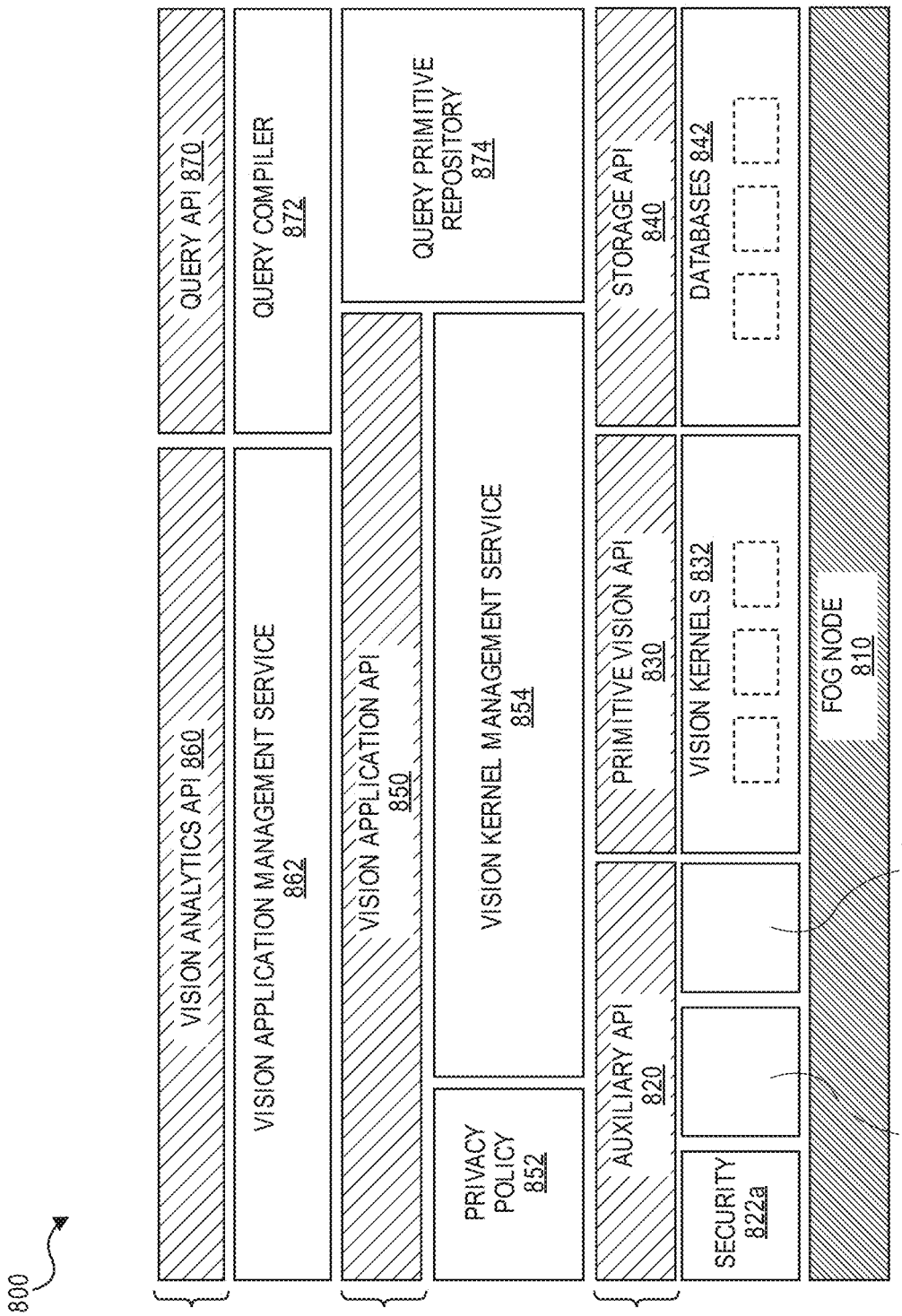

800
END-USERS/ OPERATORS
APPLICATION DEVELOPERS
ALGORITHM DEVELOPERS
VISION ANALYTICS API 860
QUERY API 870
VISION APPLICATION MANAGEMENT SERVICE 862
QUERY COMPILER 872
VISION APPLICATION API 850
QUERY PRIMITIVE REPOSITORY 874
PRIVACY POLICY 852
VISION KERNEL MANAGEMENT SERVICE 854
AUXILIARY API 820
PRIMITIVE VISION API 830
STORAGE API 840
SECURITY 822a
VISION KERNELS 832
DATABASES 842
FOG NODE 810
COMMUNICATION 822b
COMPRESSION 822c

FIG. 8

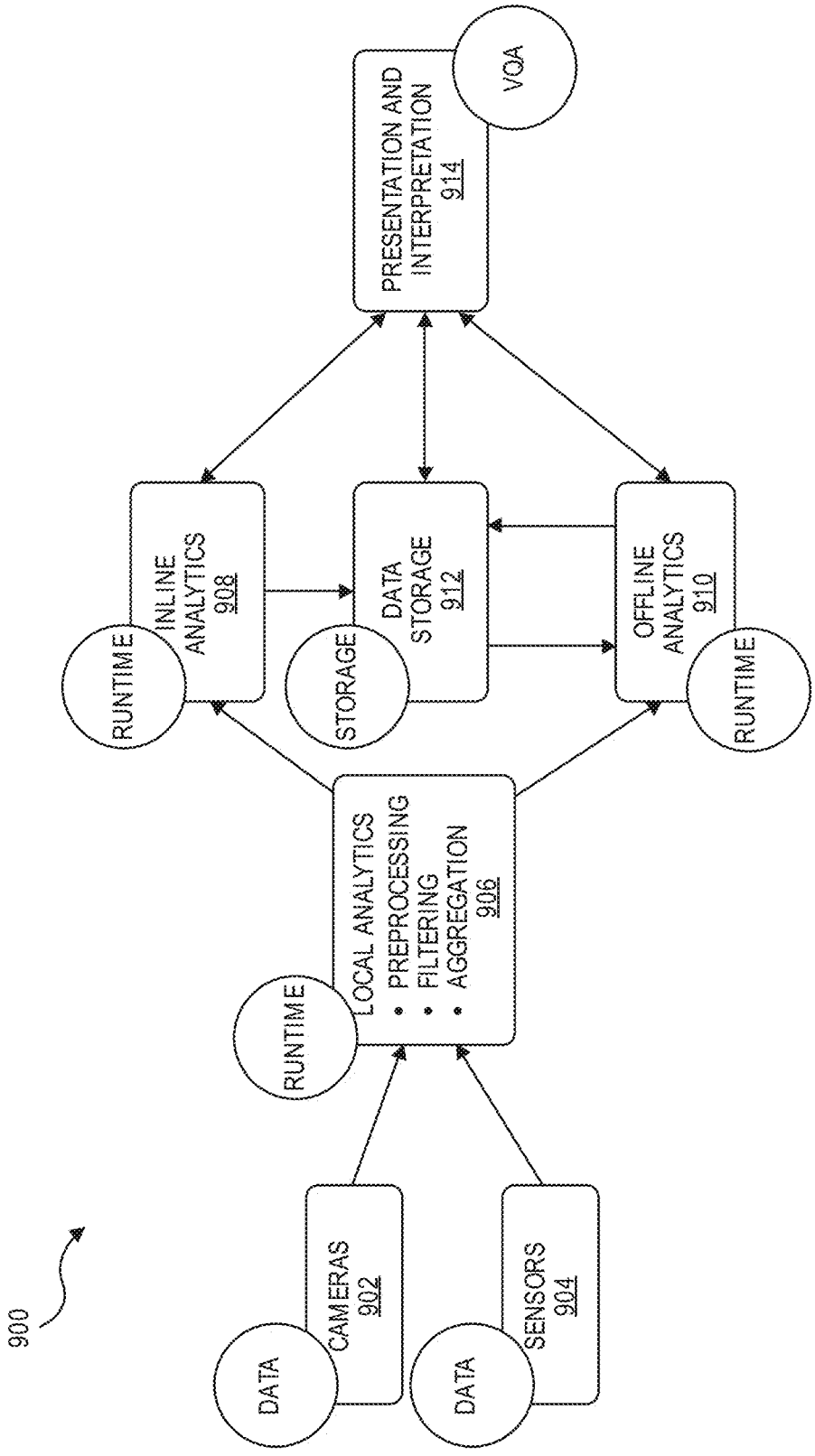

900
DATA
CAMERAS
902
DATA
SENSORS
904
RUNTIME
LOCAL ANALYTICS
• PREPROCESSING
• FILTERING
• AGGREGATION
906
RUNTIME
INLINE
ANALYTICS
908
STORAGE
DATA
STORAGE
912
RUNTIME
OFFLINE
ANALYTICS
910
PRESENTATION AND
INTERPRETATION
914
VOA

FIG. 9

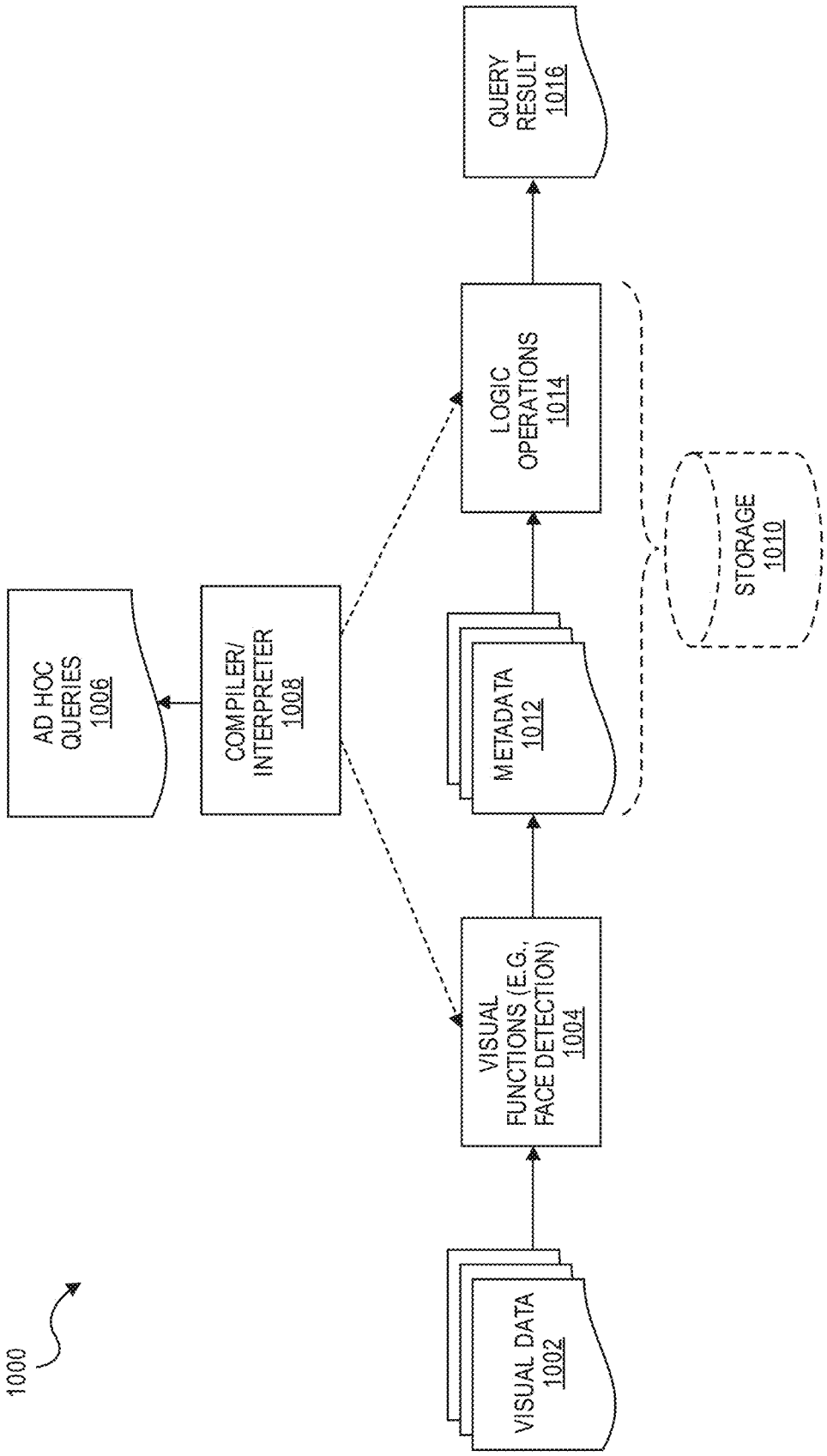

1000
VISUAL DATA 1002
VISUAL FUNCTIONS (E.G., FACE DETECTION) 1004
METADATA 1012
LOGIC OPERATIONS 1014
QUERY RESULT 1016
STORAGE 1010
COMPILER/ INTERPRETER 1008
AD HOC QUERIES 1006

FIG. 10

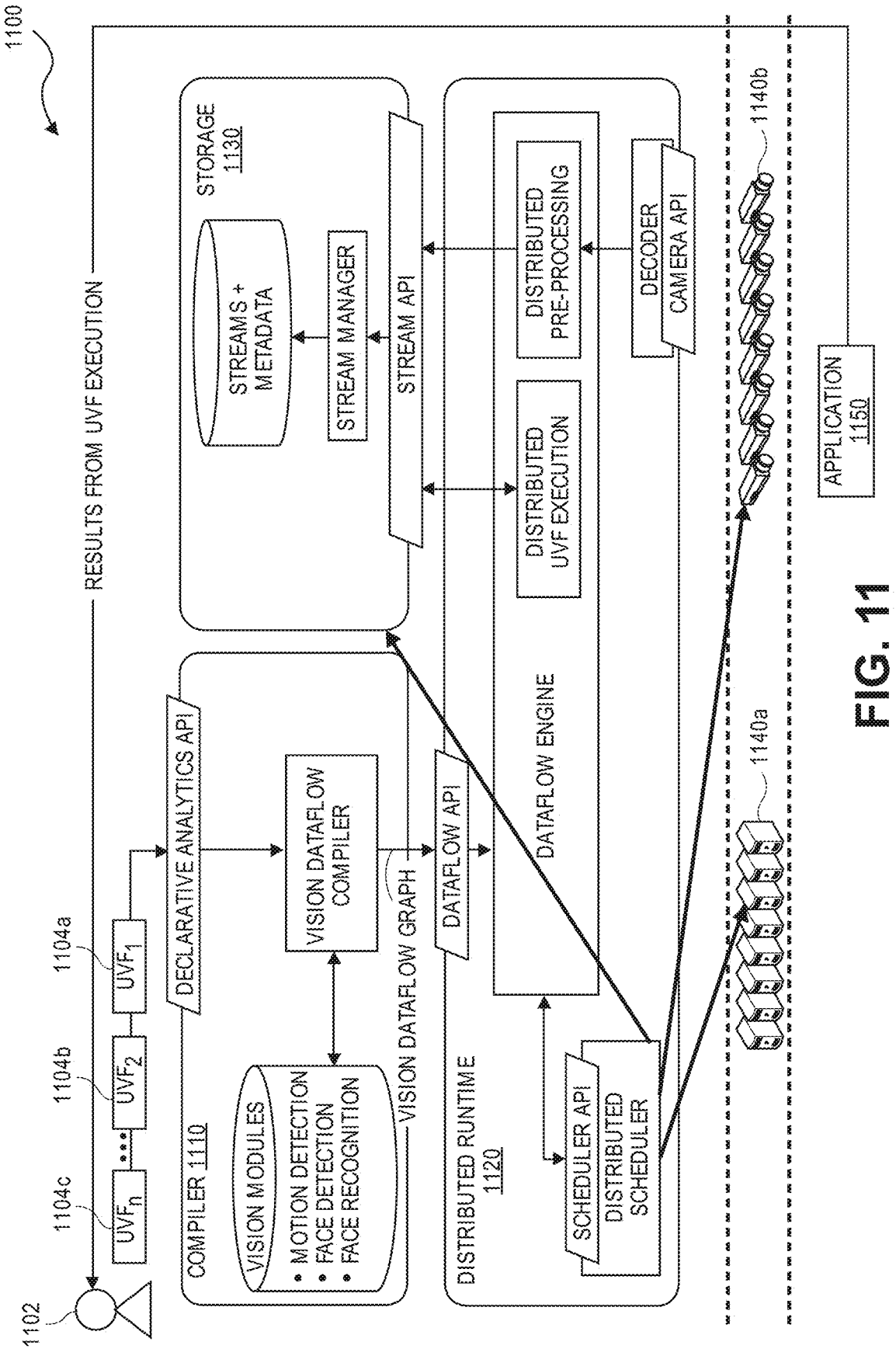

1100
RESULTS FROM UVF EXECUTION
1102
1104c
1104b
1104a
UVF$_n$
UVF$_2$
UVF$_1$
COMPILER 1110
DECLARATIVE ANALYTICS API
VISION MODULES
• MOTION DETECTION
• FACE DETECTION
• FACE RECOGNITION
VISION DATAFLOW COMPILER
VISION DATAFLOW GRAPH
STORAGE 1130
STREAMS + METADATA
STREAM MANAGER
STREAM API
DISTRIBUTED RUNTIME 1120
DATAFLOW API
DATAFLOW ENGINE
DISTRIBUTED UVF EXECUTION
DISTRIBUTED PRE-PROCESSING
DECODER
CAMERA API
SCHEDULER API
DISTRIBUTED SCHEDULER
1140a
1140b
APPLICATION 1150

FIG. 11

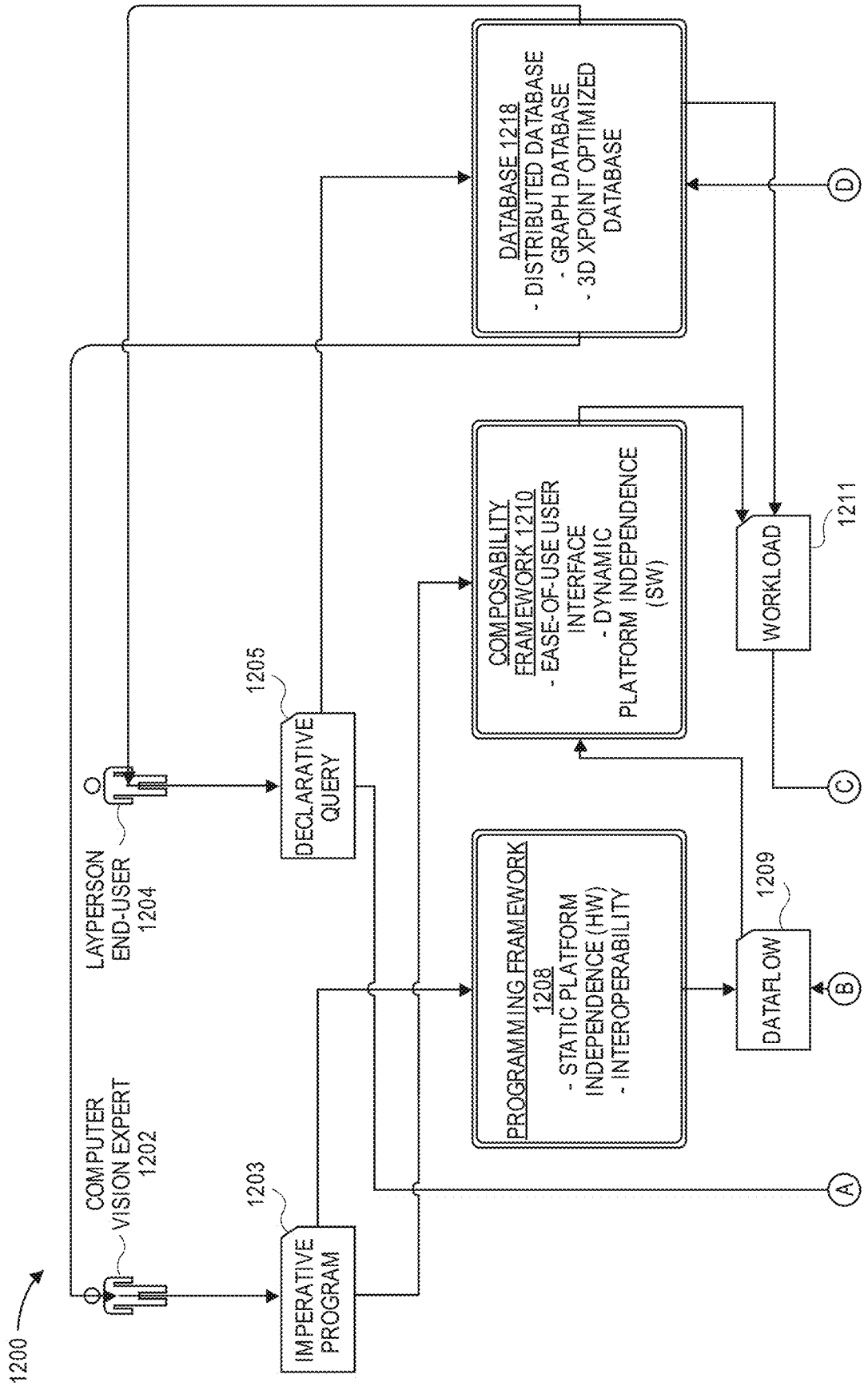

1200
COMPUTER VISION EXPERT 1202
LAYPERSON END-USER 1204
1203
IMPERATIVE PROGRAM
1205
DECLARATIVE QUERY
PROGRAMMING FRAMEWORK 1208
- STATIC PLATFORM INDEPENDENCE (HW)
- INTEROPERABILITY
COMPOSABILITY FRAMEWORK 1210
- EASE-OF-USE USER INTERFACE
- DYNAMIC PLATFORM INDEPENDENCE (SW)
DATABASE 1218
- DISTRIBUTED DATABASE
- GRAPH DATABASE
- 3D XPOINT OPTIMIZED DATABASE
DATAFLOW
1209
WORKLOAD
1211
A
B
C
D

FIG. 12A

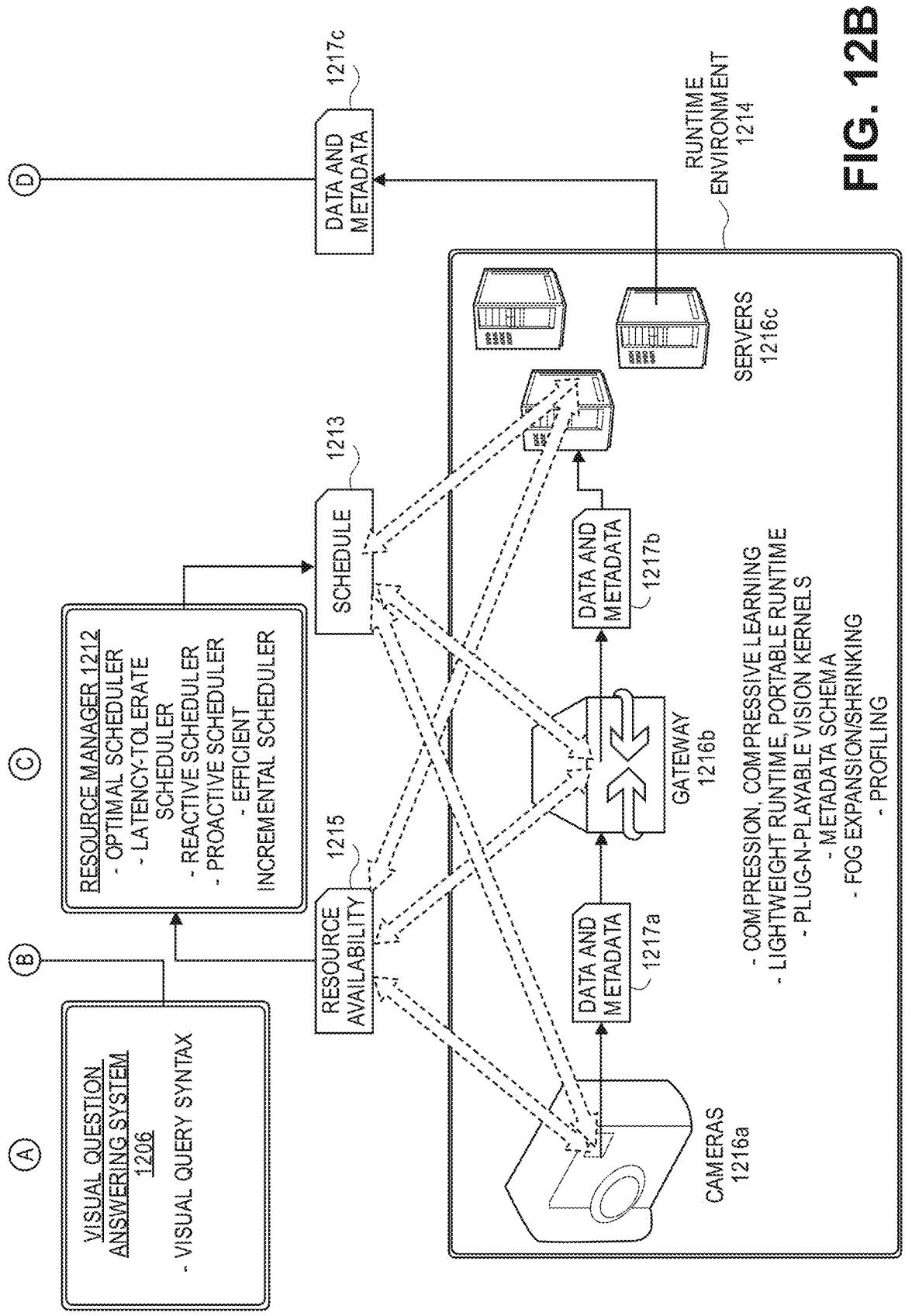

A
VISUAL QUESTION ANSWERING SYSTEM 1206
- VISUAL QUERY SYNTAX
B
C
RESOURCE MANAGER 1212
- OPTIMAL SCHEDULER
- LATENCY-TOLERATE SCHEDULER
- REACTIVE SCHEDULER
- PROACTIVE SCHEDULER
- EFFICIENT INCREMENTAL SCHEDULER
RESOURCE AVAILABILITY 1215
SCHEDULE 1213
D
DATA AND METADATA 1217c
RUNTIME ENVIRONMENT 1214
SERVERS 1216c
DATA AND METADATA 1217b
GATEWAY 1216b
DATA AND METADATA 1217a
- COMPRESSION, COMPRESSIVE LEARNING
- LIGHTWEIGHT RUNTIME, PORTABLE RUNTIME
- PLUG-N-PLAYABLE VISION KERNELS
- METADATA SCHEMA
- FOG EXPANSION/SHRINKING
- PROFILING
CAMERAS 1216a

FIG. 12B

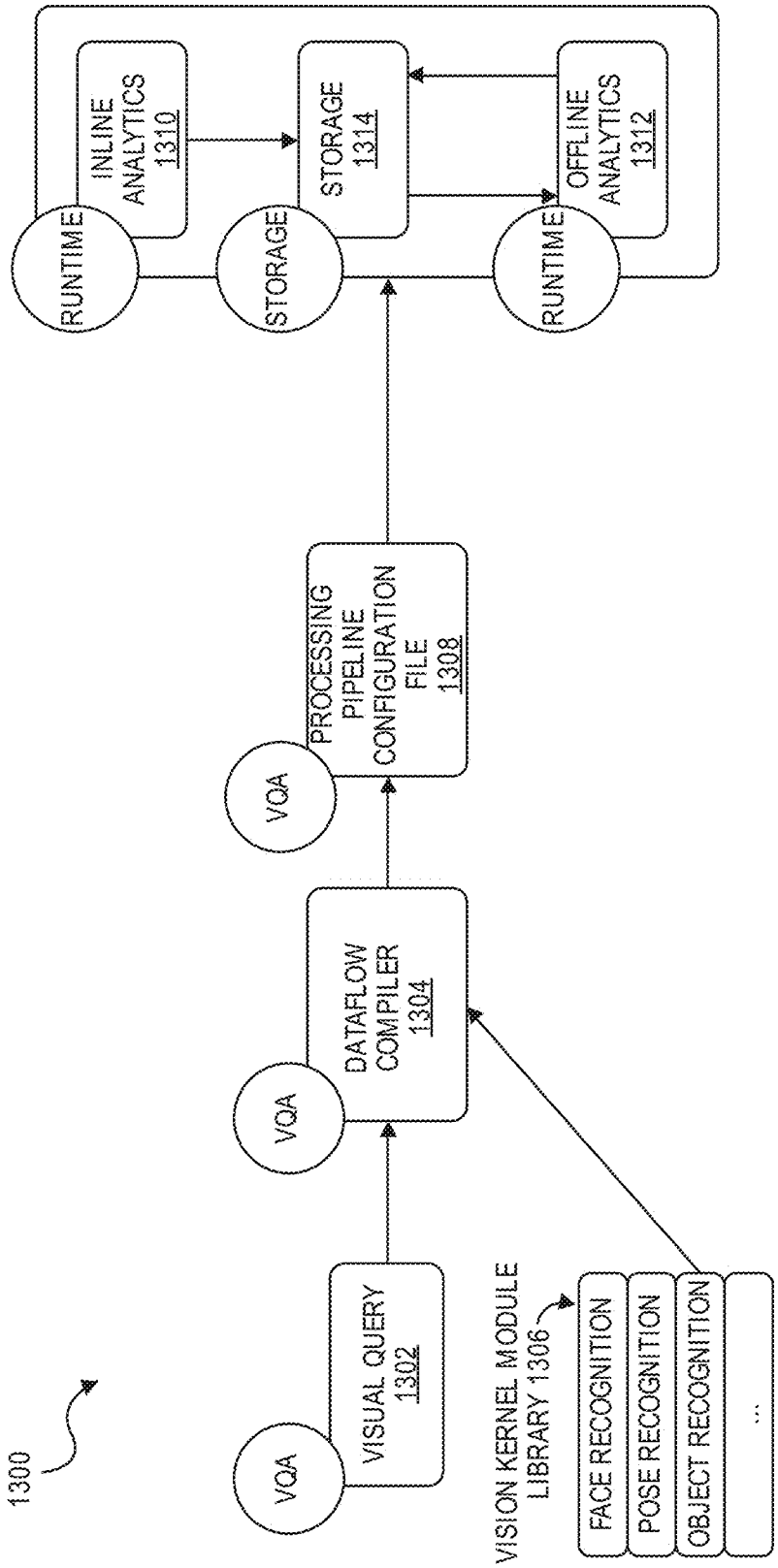

1300
VQA
VISUAL QUERY 1302
VQA
DATAFLOW COMPILER 1304
VQA
PROCESSING PIPELINE CONFIGURATION FILE 1308
VISION KERNEL MODULE LIBRARY 1306
FACE RECOGNITION
POSE RECOGNITION
OBJECT RECOGNITION
...
RUNTIME
INLINE ANALYTICS 1310
STORAGE
STORAGE 1314
RUNTIME
OFFLINE ANALYTICS 1312

FIG. 13

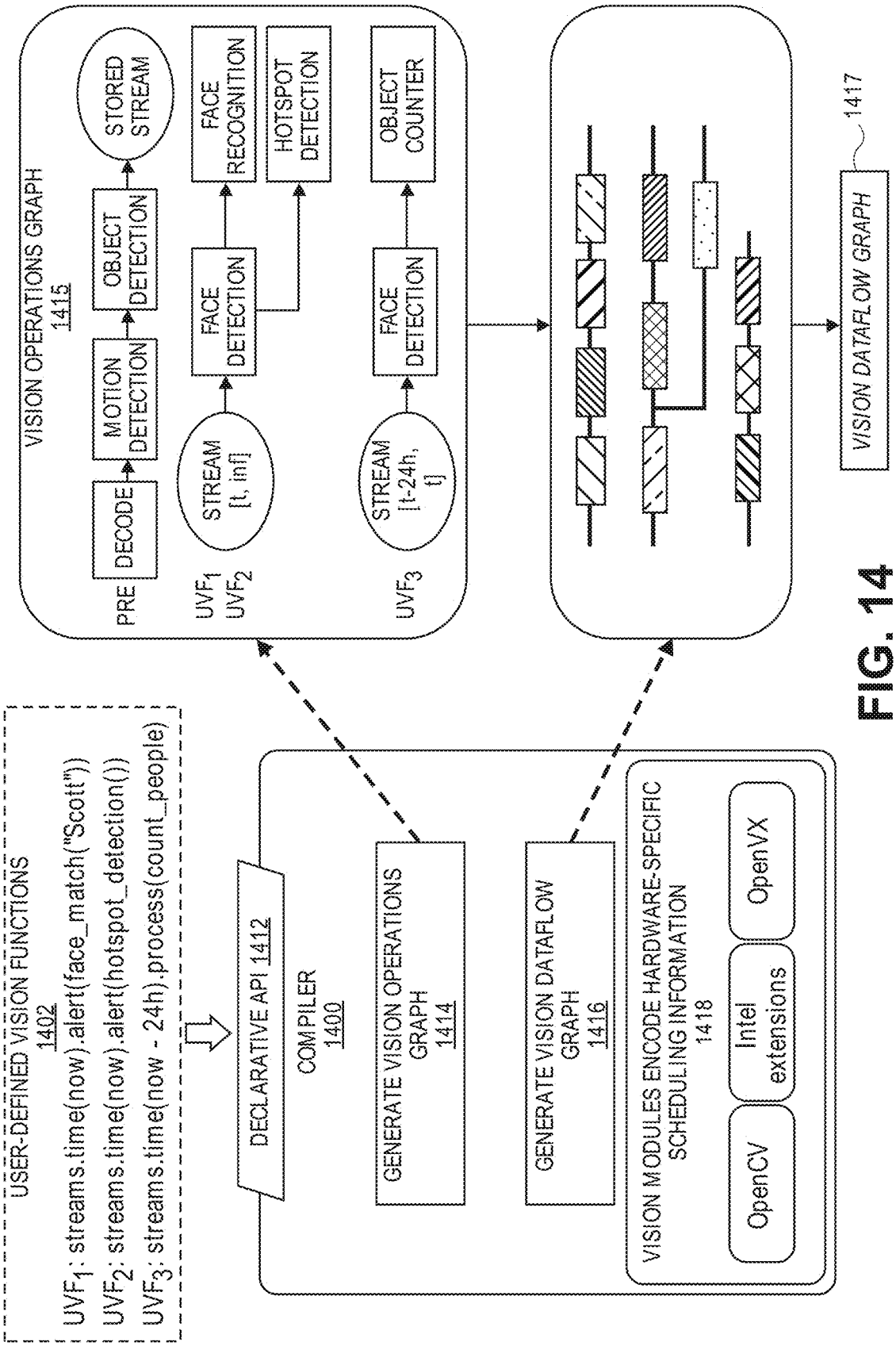

USER-DEFINED VISION FUNCTIONS
1402
UVF1: streams.time(now).alert(face_match("Scott"))
UVF2: streams.time(now).alert(hotspot_detection())
UVF3: streams.time(now - 24h).process(count_people)
DECLARATIVE API 1412
COMPILER
1400
GENERATE VISION OPERATIONS GRAPH
1414
GENERATE VISION DATAFLOW GRAPH
1416
VISION MODULES ENCODE HARDWARE-SPECIFIC SCHEDULING INFORMATION
1418
OpenCV
Intel extensions
OpenVX
VISION OPERATIONS GRAPH
1415
PRE
DECODE
MOTION DETECTION
OBJECT DETECTION
STORED STREAM
UVF1
UVF2
STREAM [t, inf]
FACE DETECTION
FACE RECOGNITION
HOTSPOT DETECTION
UVF3
STREAM [t-24h, t]
FACE DETECTION
OBJECT COUNTER
VISION DATAFLOW GRAPH
1417

FIG. 14

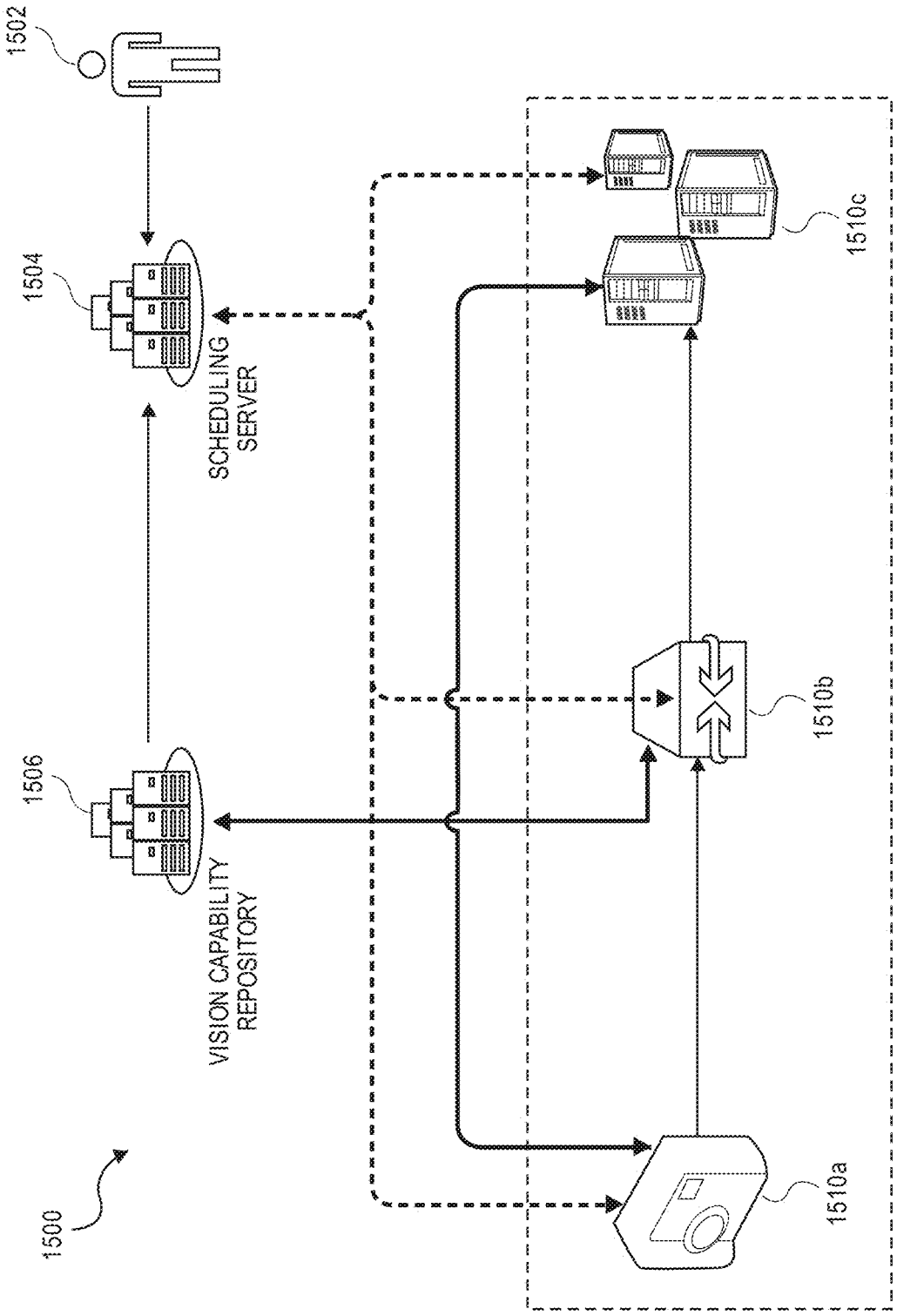

1502
1504
SCHEDULING SERVER
1506
VISION CAPABILITY REPOSITORY
1500
1510a
1510b
1510c

FIG. 15

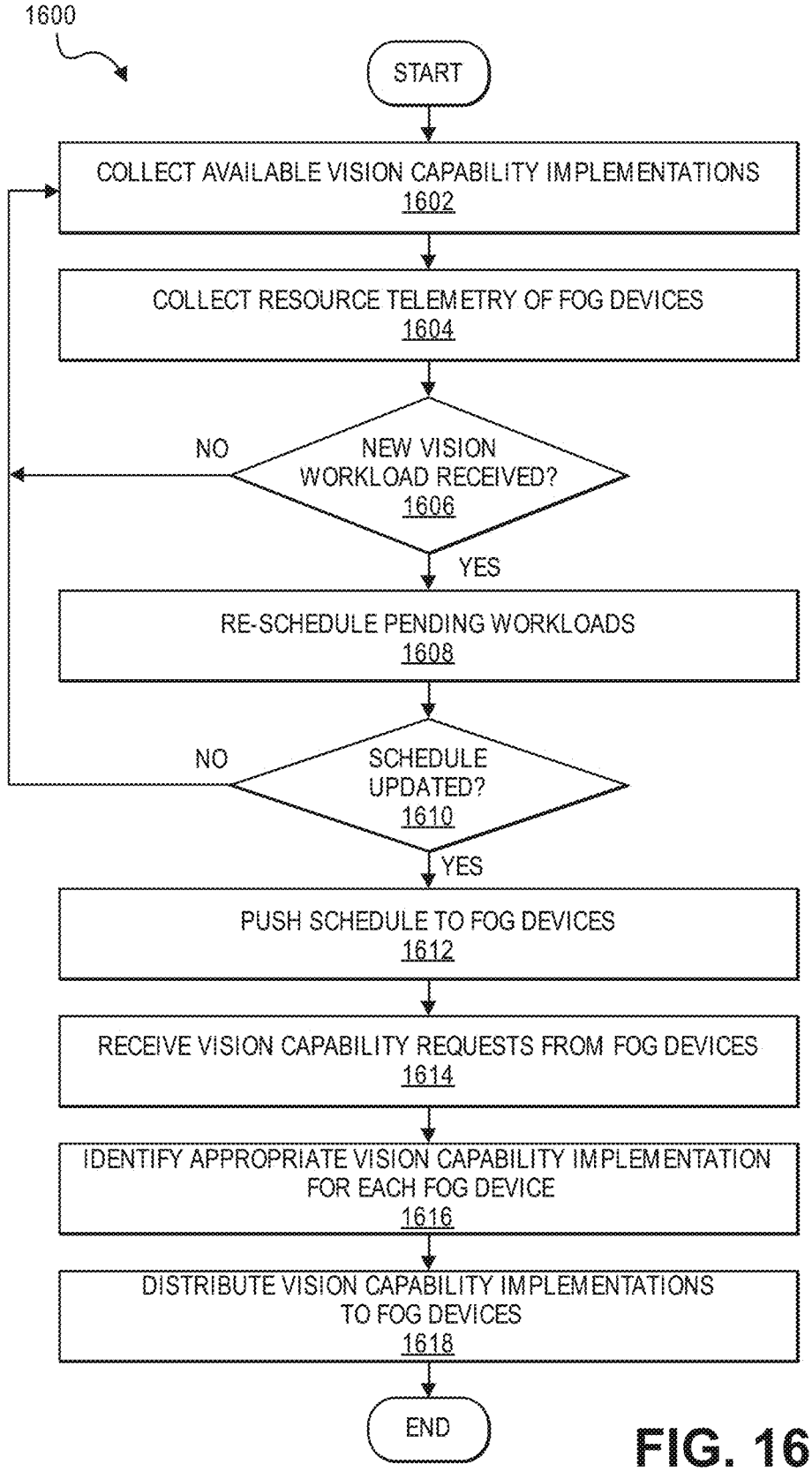

1600
START
COLLECT AVAILABLE VISION CAPABILITY IMPLEMENTATIONS
1602
COLLECT RESOURCE TELEMETRY OF FOG DEVICES
1604
NEW VISION WORKLOAD RECEIVED?
1606
NO
YES
RE-SCHEDULE PENDING WORKLOADS
1608
SCHEDULE UPDATED?
1610
NO
YES
PUSH SCHEDULE TO FOG DEVICES
1612
RECEIVE VISION CAPABILITY REQUESTS FROM FOG DEVICES
1614
IDENTIFY APPROPRIATE VISION CAPABILITY IMPLEMENTATION FOR EACH FOG DEVICE
1616
DISTRIBUTE VISION CAPABILITY IMPLEMENTATIONS TO FOG DEVICES
1618
END

FIG. 16

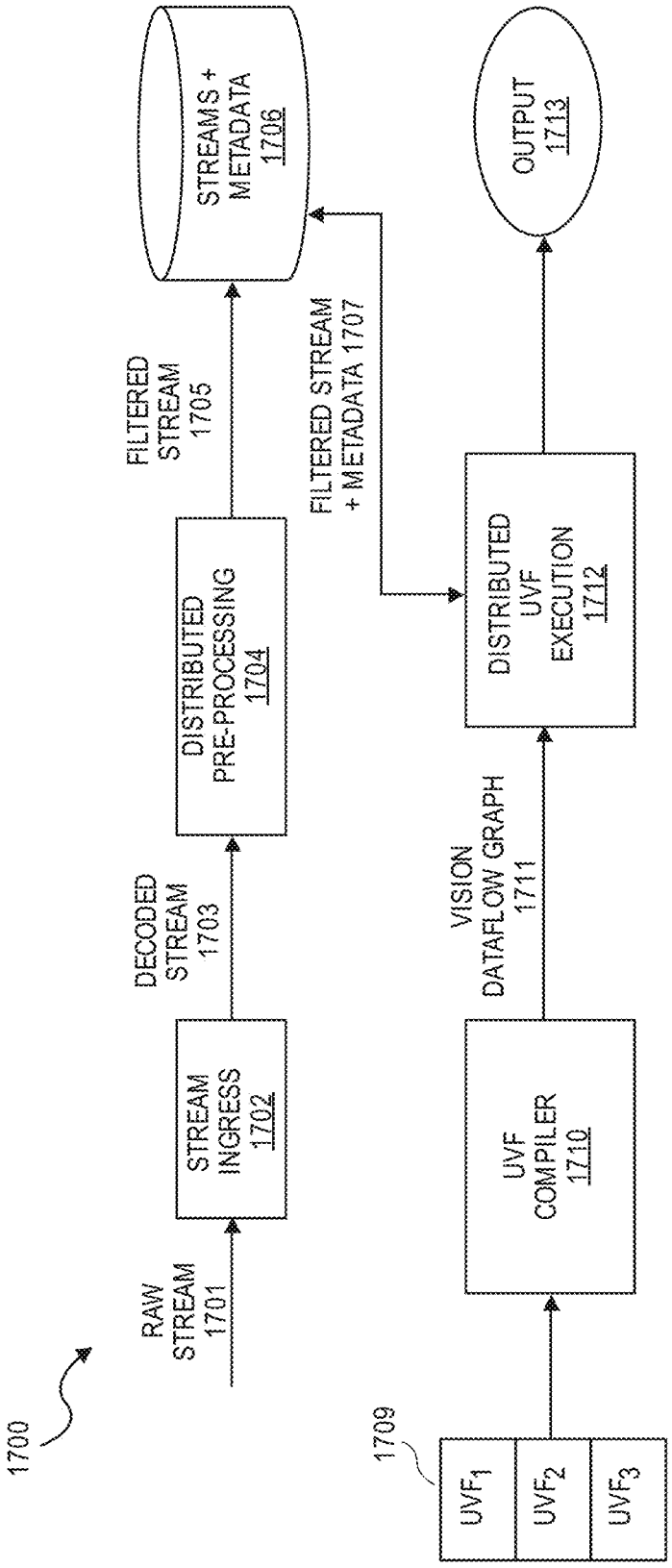

1700
RAW STREAM 1701
STREAM INGRESS 1702
DECODED STREAM 1703
DISTRIBUTED PRE-PROCESSING 1704
FILTERED STREAM 1705
STREAMS + METADATA 1706
1709
UVF1
UVF2
UVF3
UVF COMPILER 1710
VISION DATAFLOW GRAPH 1711
DISTRIBUTED UVF EXECUTION 1712
FILTERED STREAM + METADATA 1707
OUTPUT 1713

FIG. 17

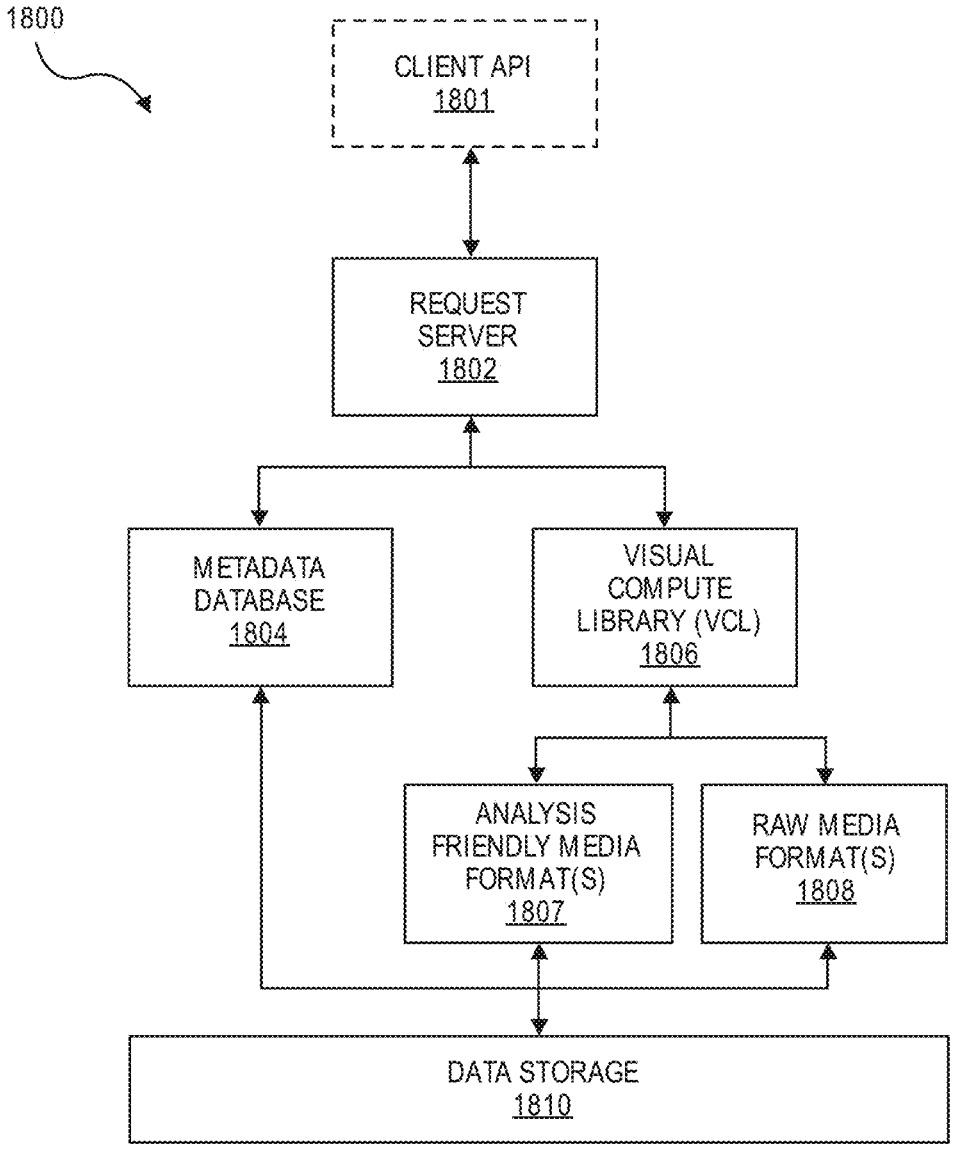

1800
CLIENT API
1801
REQUEST
SERVER
1802
METADATA
DATABASE
1804
VISUAL
COMPUTE
LIBRARY (VCL)
1806
ANALYSIS
FRIENDLY MEDIA
FORMAT(S)
1807
RAW MEDIA
FORMAT(S)
1808
DATA STORAGE
1810

FIG. 18

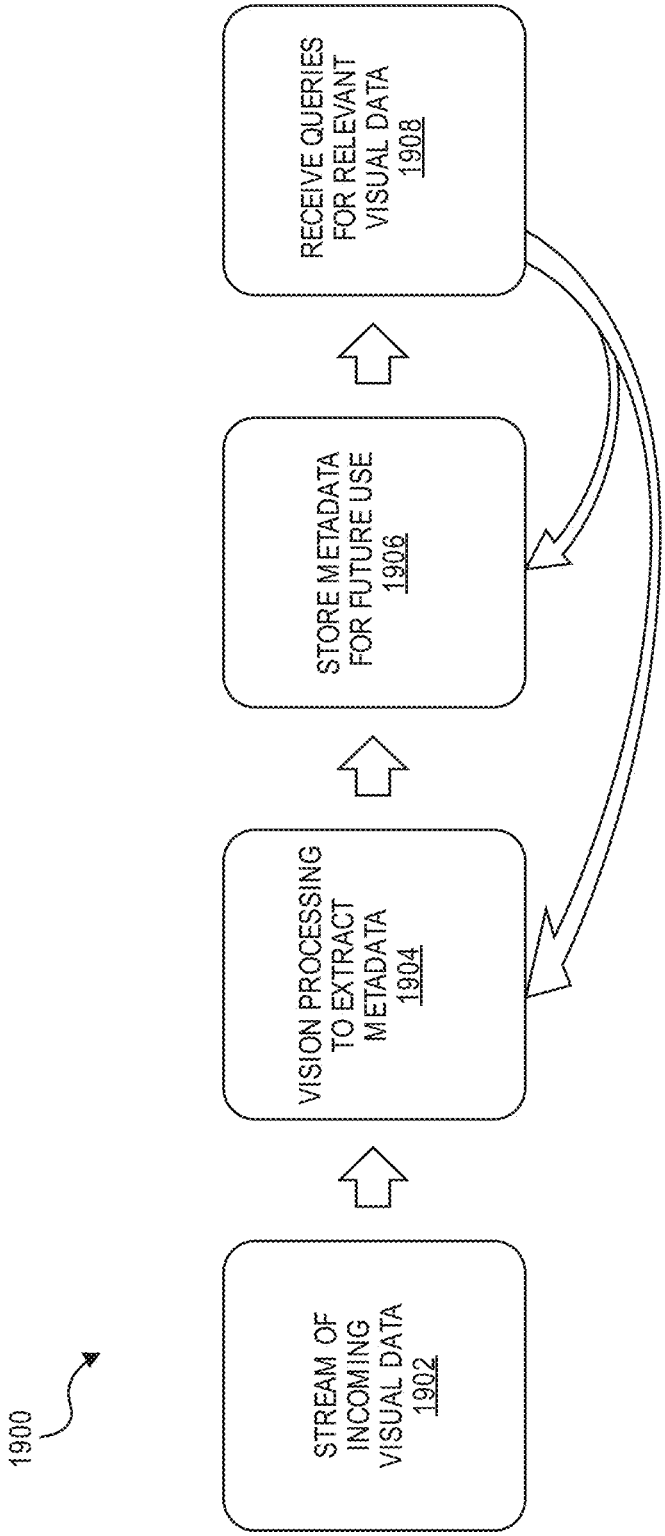

1900
STREAM OF INCOMING VISUAL DATA 1902
VISION PROCESSING TO EXTRACT METADATA 1904
STORE METADATA FOR FUTURE USE 1906
RECEIVE QUERIES FOR RELEVANT VISUAL DATA 1908

FIG. 19

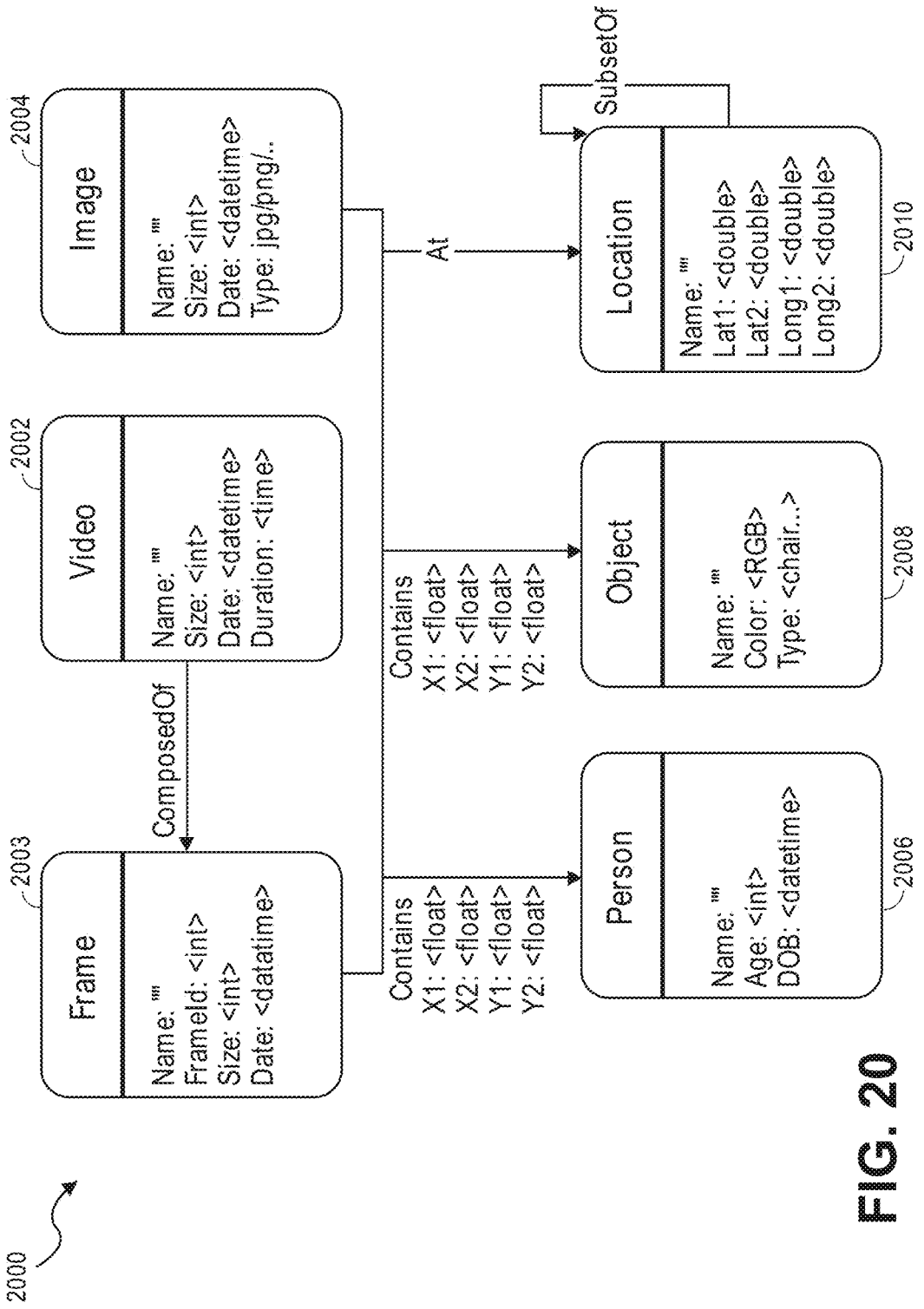

2000
2003
Frame
Name: ""
FrameId: <int>
Size: <int>
Date: <datatime>
ComposedOf
2002
Video
Name: ""
Size: <int>
Date: <datetime>
Duration: <time>
2004
Image
Name: ""
Size: <int>
Date: <datetime>
Type: jpg/png/..
Contains
X1: <float>
X2: <float>
Y1: <float>
Y2: <float>
Person
Name: ""
Age: <int>
DOB: <datetime>
2006
Contains
X1: <float>
X2: <float>
Y1: <float>
Y2: <float>
Object
Name: ""
Color: <RGB>
Type: <chair...>
2008
At
SubsetOf
Location
Name: ""
Lat1: <double>
Lat2: <double>
Long1: <double>
Long2: <double>
2010

FIG. 20

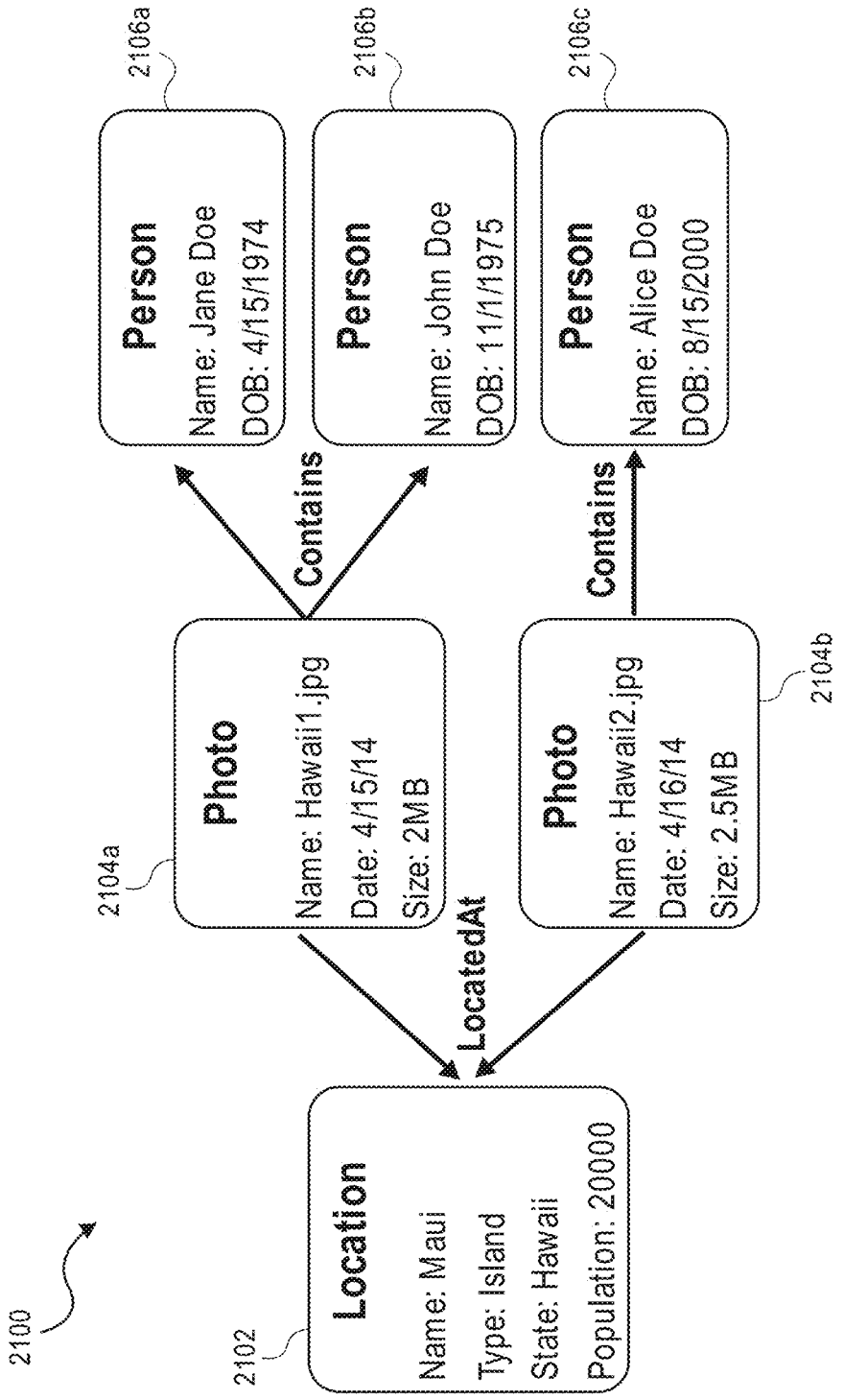

2100
2102
Location
Name: Maui
Type: Island
State: Hawaii
Population: 20000
2104a
Photo
Name: Hawaii1.jpg
Date: 4/15/14
Size: 2MB
2104b
Photo
Name: Hawaii2.jpg
Date: 4/16/14
Size: 2.5MB
LocatedAt
Contains
Contains
2106a
Person
Name: Jane Doe
DOB: 4/15/1974
2106b
Person
Name: John Doe
DOB: 11/1/1975
2106c
Person
Name: Alice Doe
DOB: 8/15/2000

FIG. 21

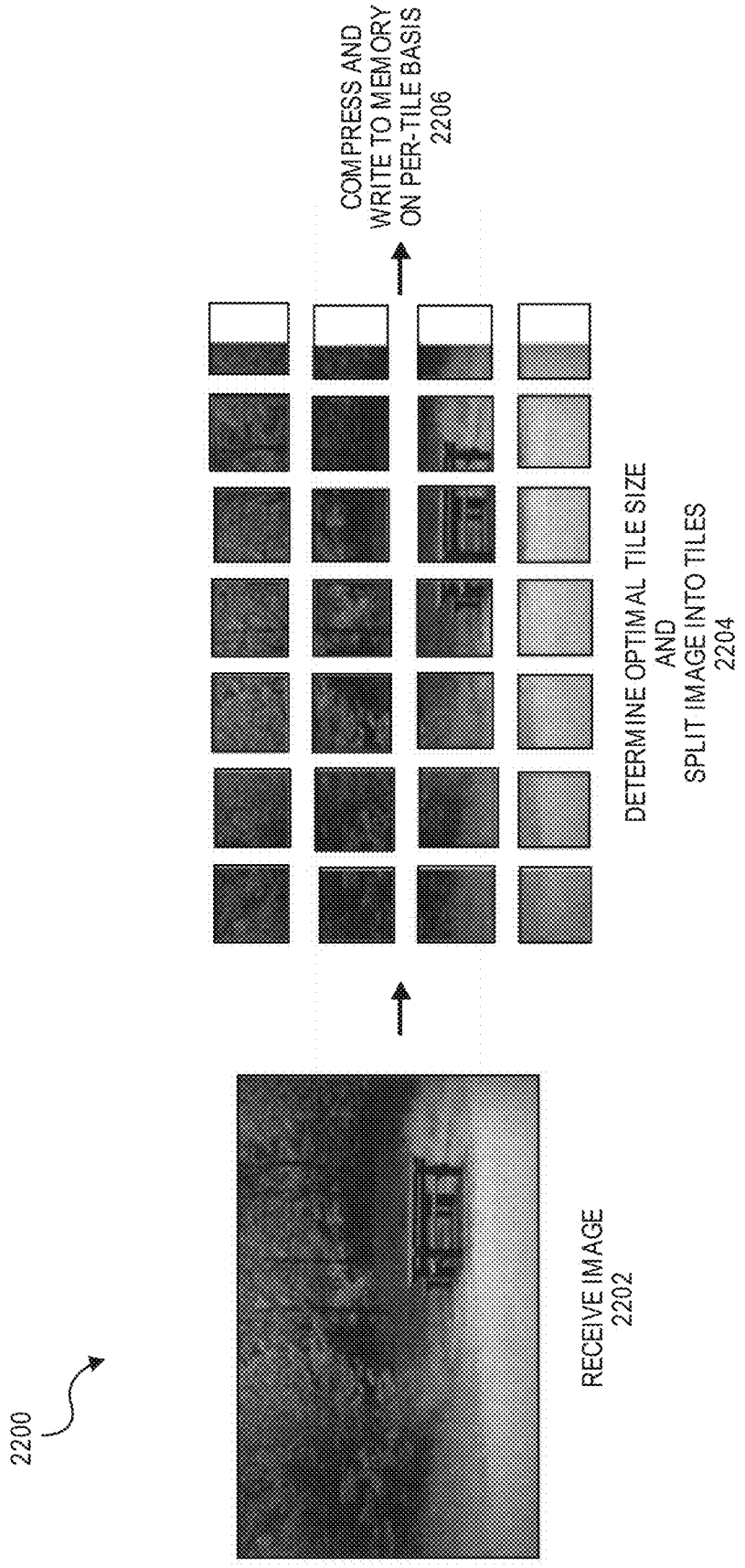

2200
RECEIVE IMAGE
2202
DETERMINE OPTIMAL TILE SIZE
AND
SPLIT IMAGE INTO TILES
2204
COMPRESS AND
WRITE TO MEMORY
ON PER-TILE BASIS
2206

FIG. 22

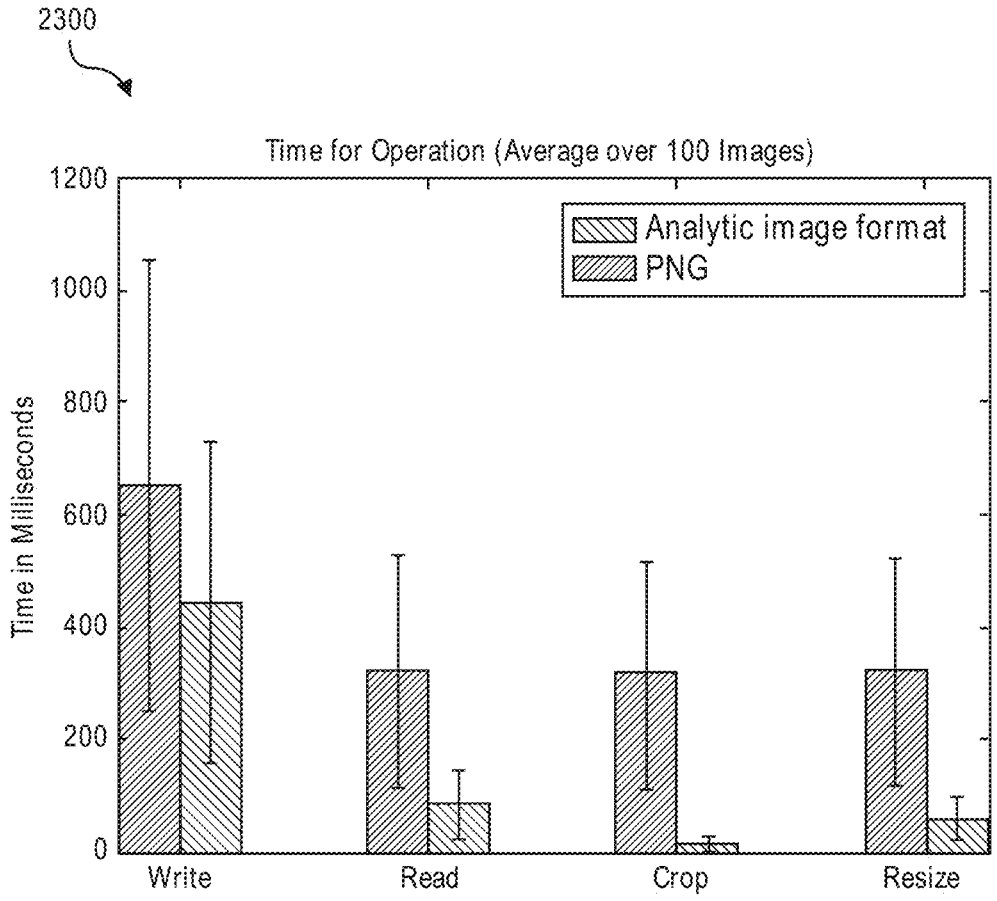

2300
Time for Operation (Average over 100 Images)
Analytic image format
PNG
Time in Milliseconds
1200
1000
800
600
400
200
0
Write
Read
Crop
Resize

FIG. 23

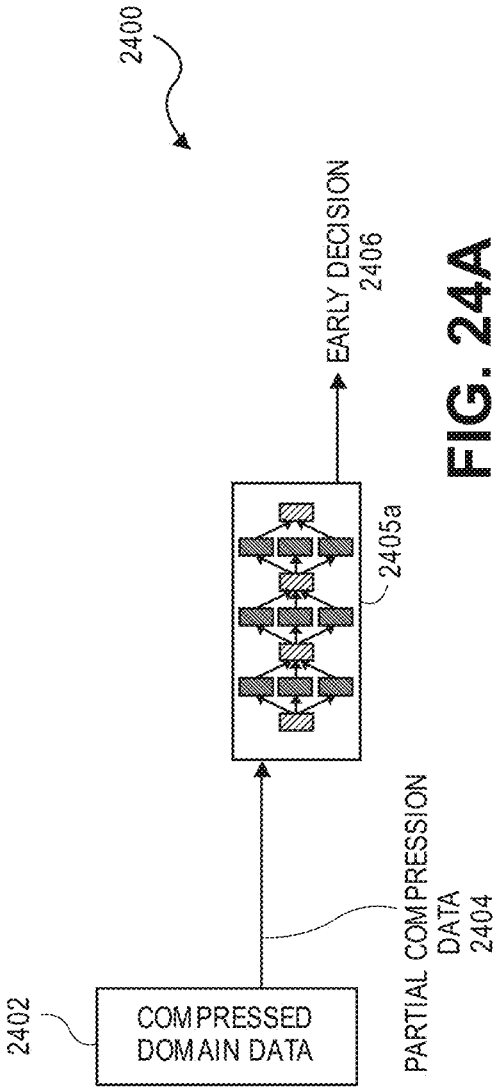


FIG. 24A

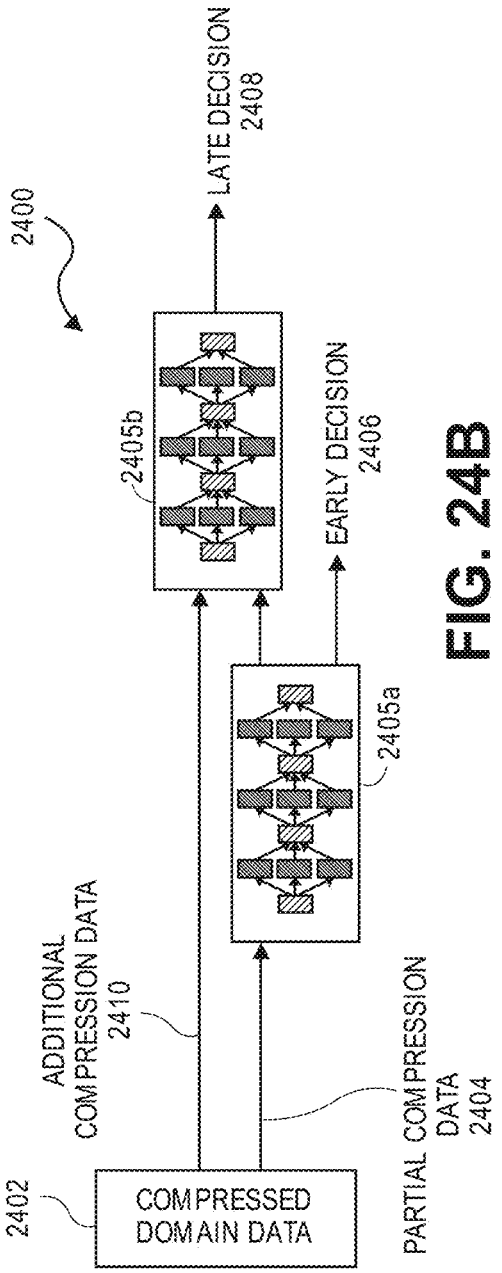


FIG. 24B

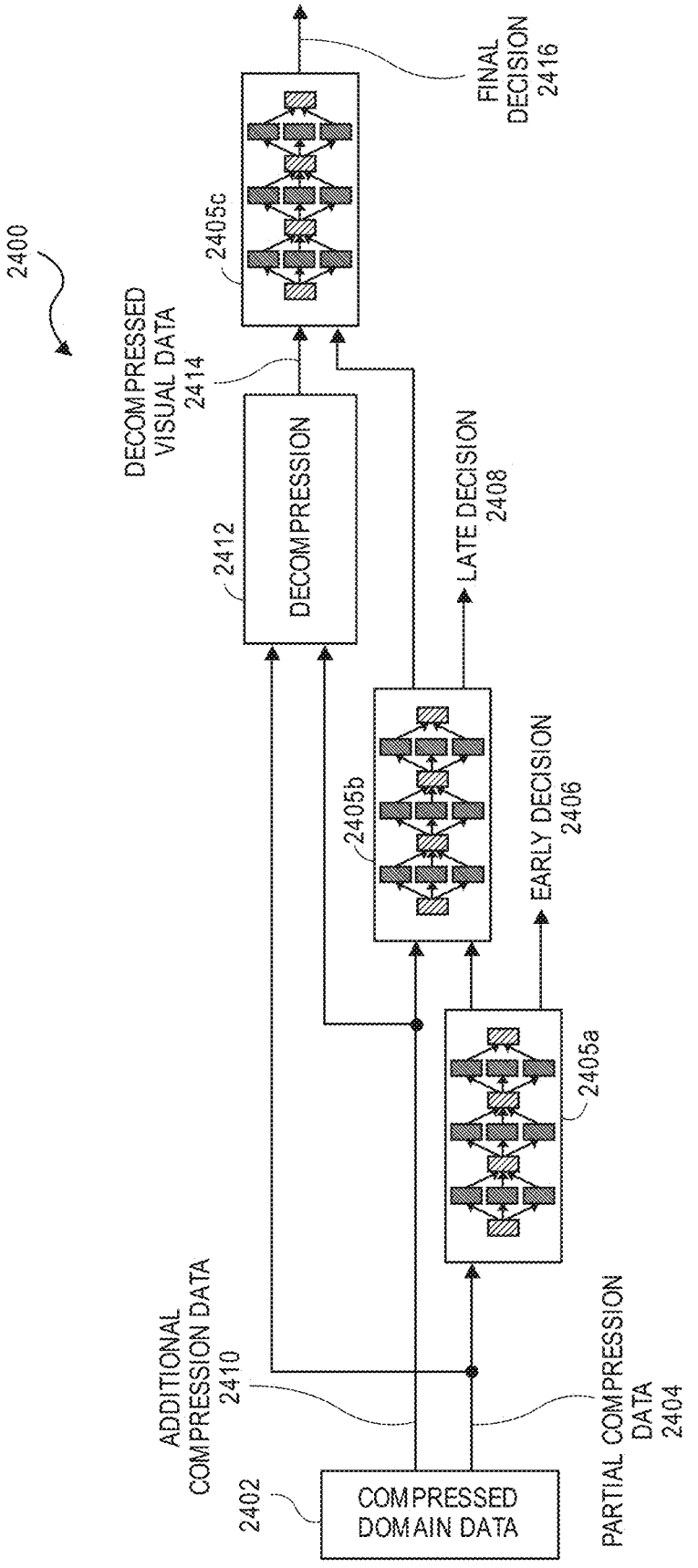

2400
COMPRESSED DOMAIN DATA
2402
ADDITIONAL COMPRESSION DATA
2410
PARTIAL COMPRESSION DATA
2404
2405a
EARLY DECISION
2406
2405b
LATE DECISION
2408
2412
DECOMPRESSION
DECOMPRESSED VISUAL DATA
2414
2405c
FINAL DECISION
2416

FIG. 24C

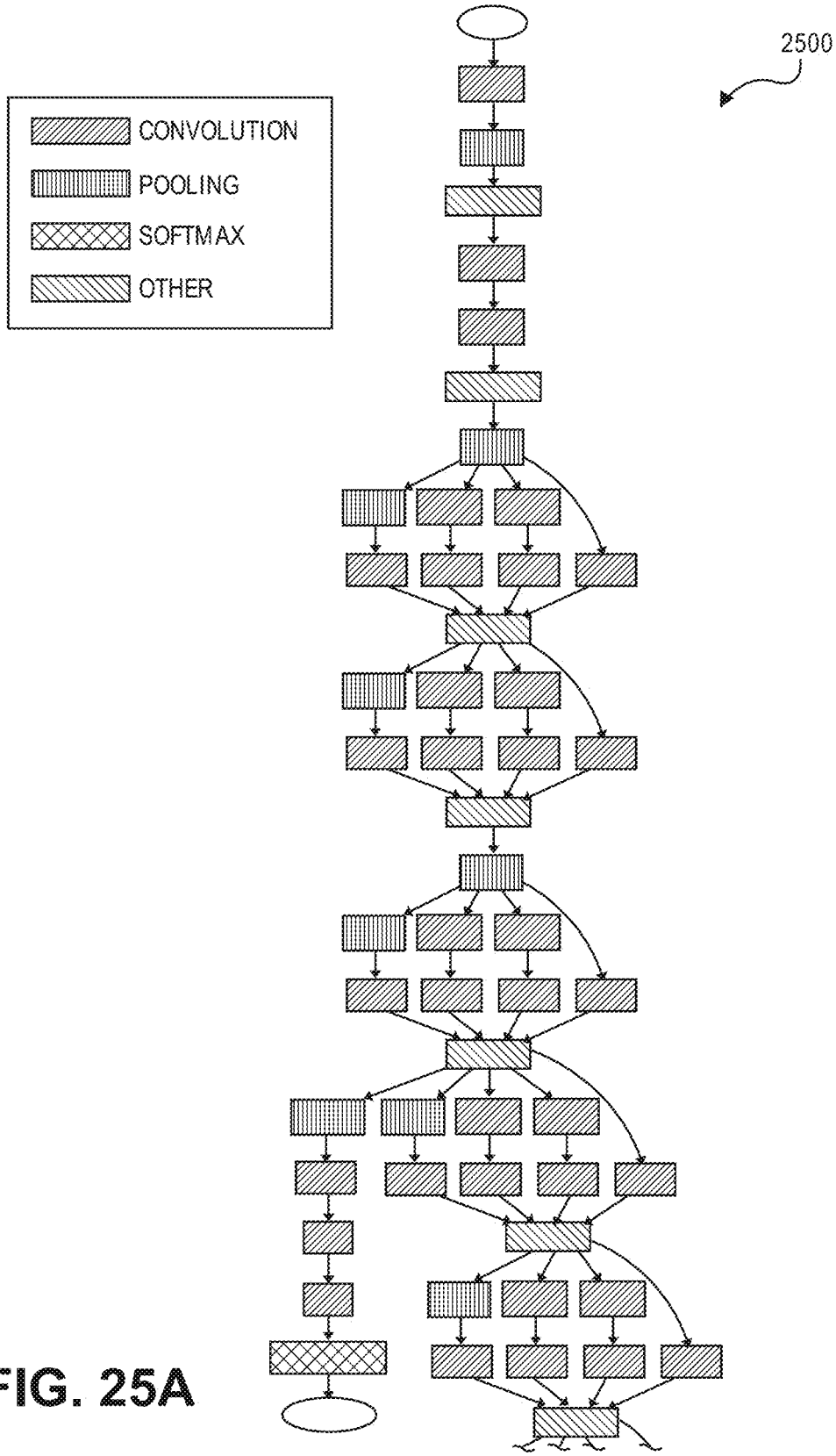


FIG. 25A

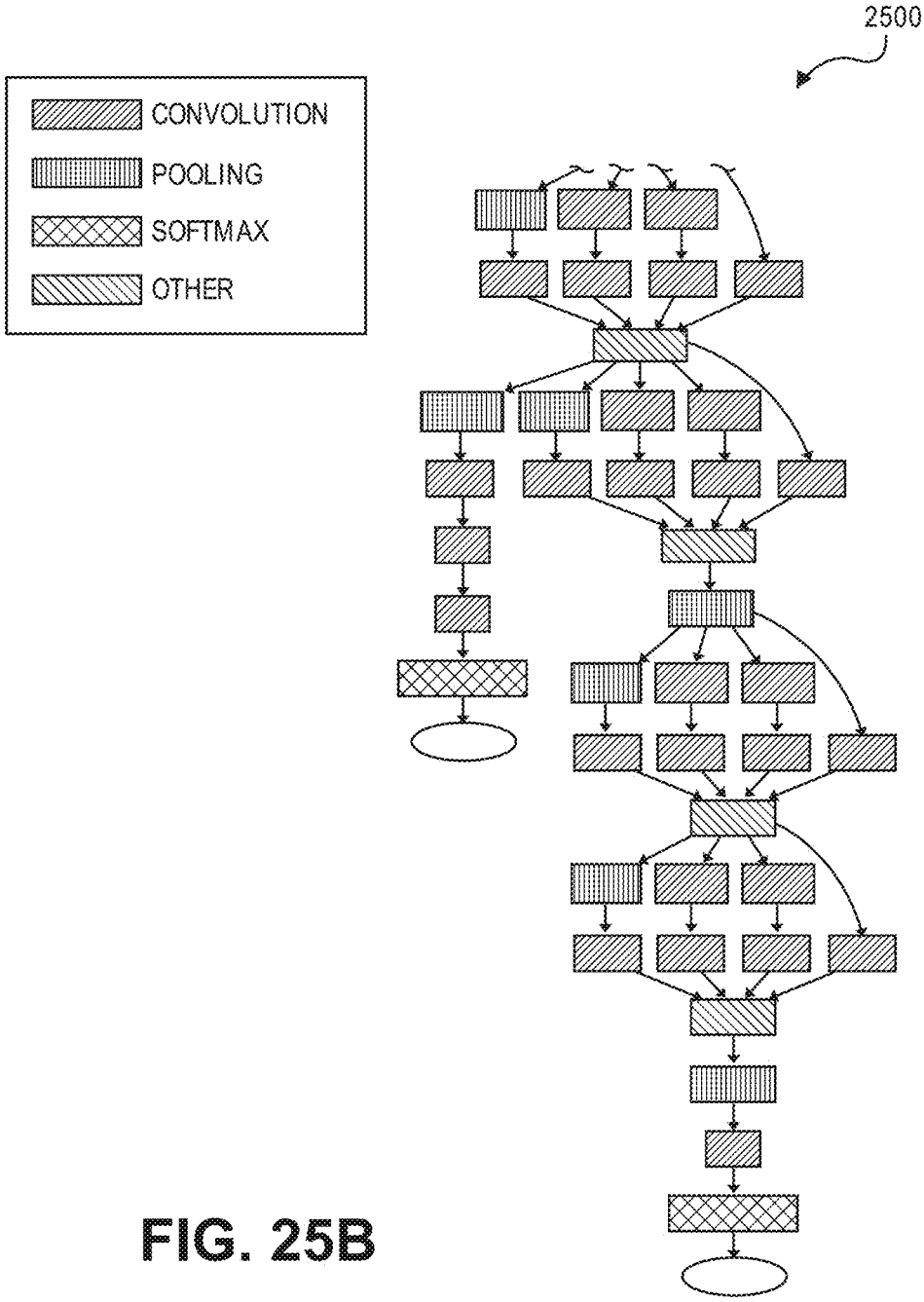


FIG. 25B

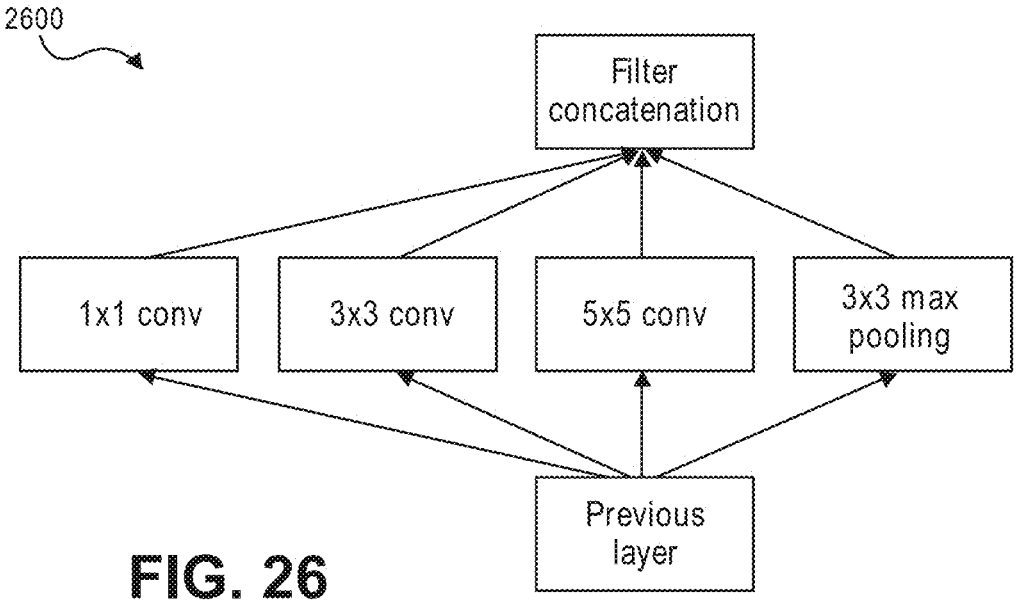

2600
Filter concatenation
1x1 conv
3x3 conv
5x5 conv
3x3 max pooling
Previous layer

FIG. 26

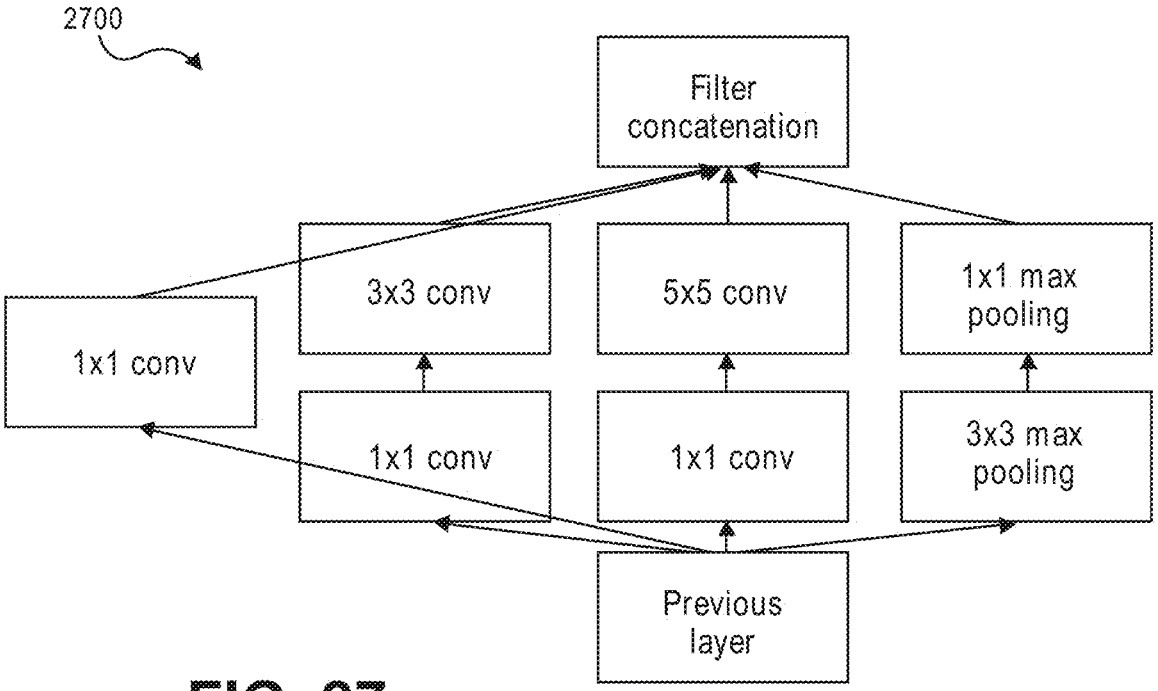

2700
Filter concatenation
1x1 conv
3x3 conv
5x5 conv
1x1 max pooling
1x1 conv
1x1 conv
3x3 max pooling
Previous layer

FIG. 27

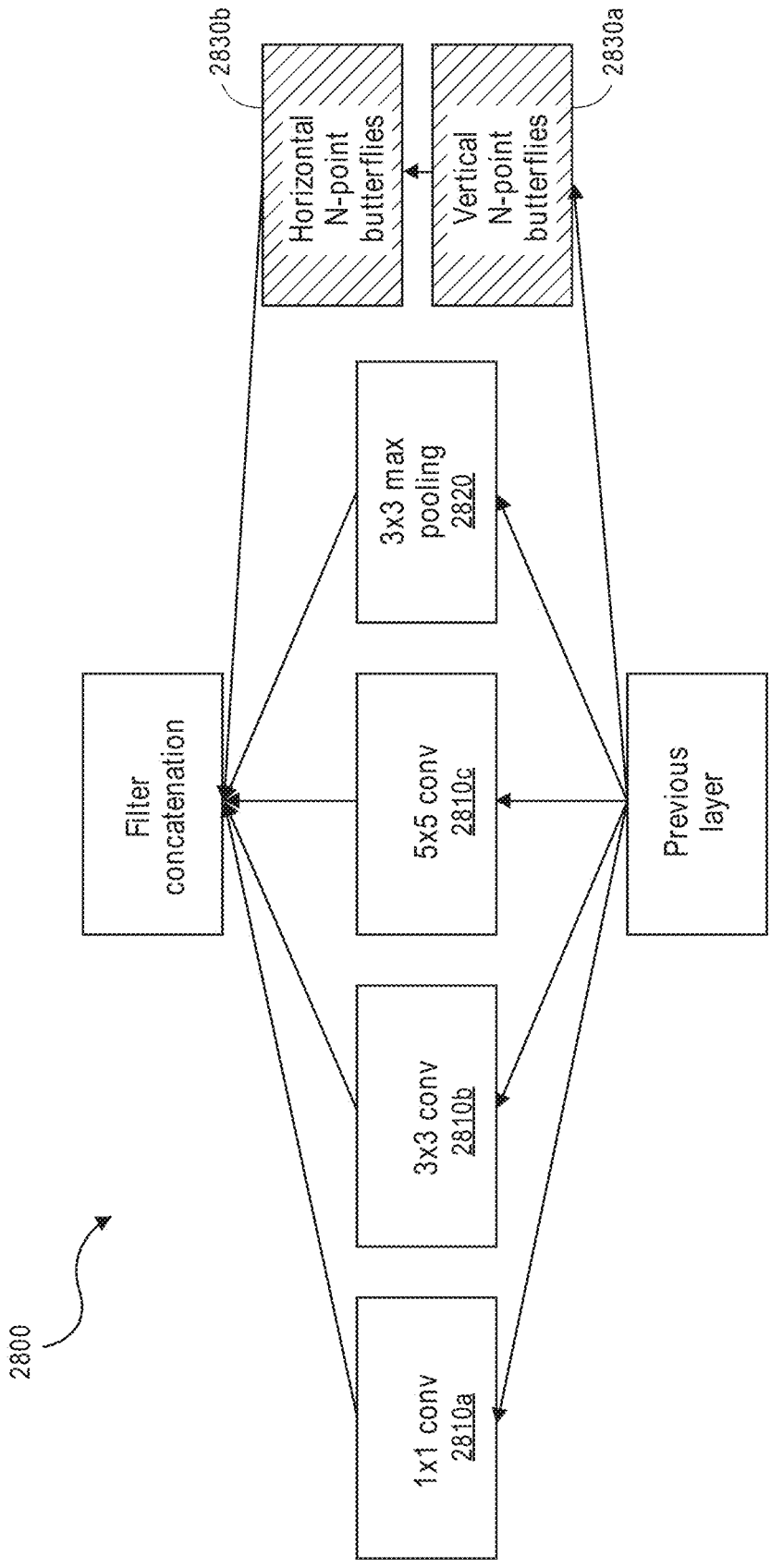

2800
1x1 conv
2810a
3x3 conv
2810b
5x5 conv
2810c
3x3 max
pooling
2820
Filter
concatenation
Previous
layer
Horizontal
N-point
butterflies
2830b
Vertical
N-point
butterflies
2830a

FIG. 28

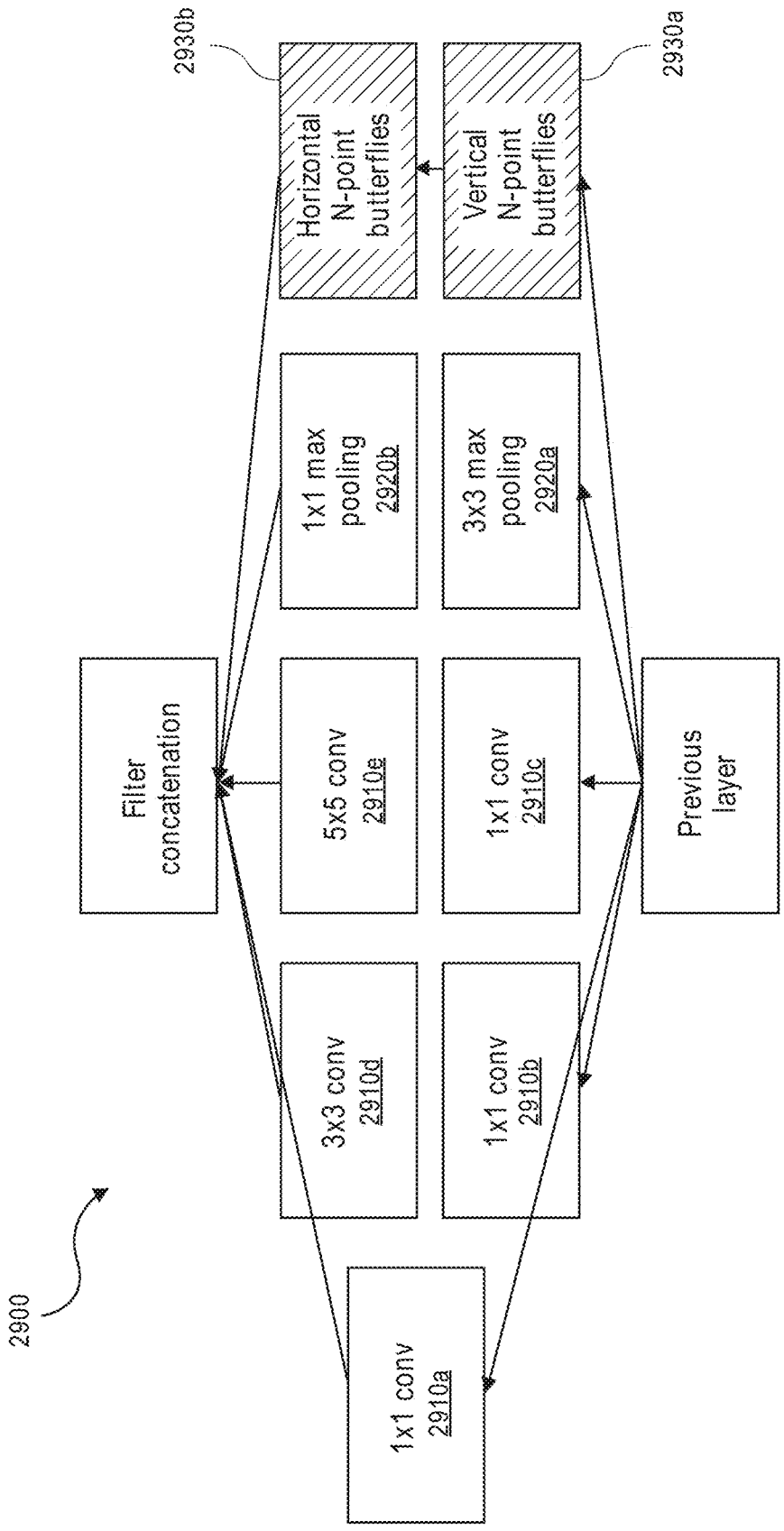

2900
1x1 conv 2910a
3x3 conv 2910d
1x1 conv 2910b
5x5 conv 2910e
1x1 conv 2910c
1x1 max pooling 2920b
3x3 max pooling 2920a
Horizontal N-point butterflies
Vertical N-point butterflies
2930b
2930a
Filter concatenation
Previous layer

FIG. 29

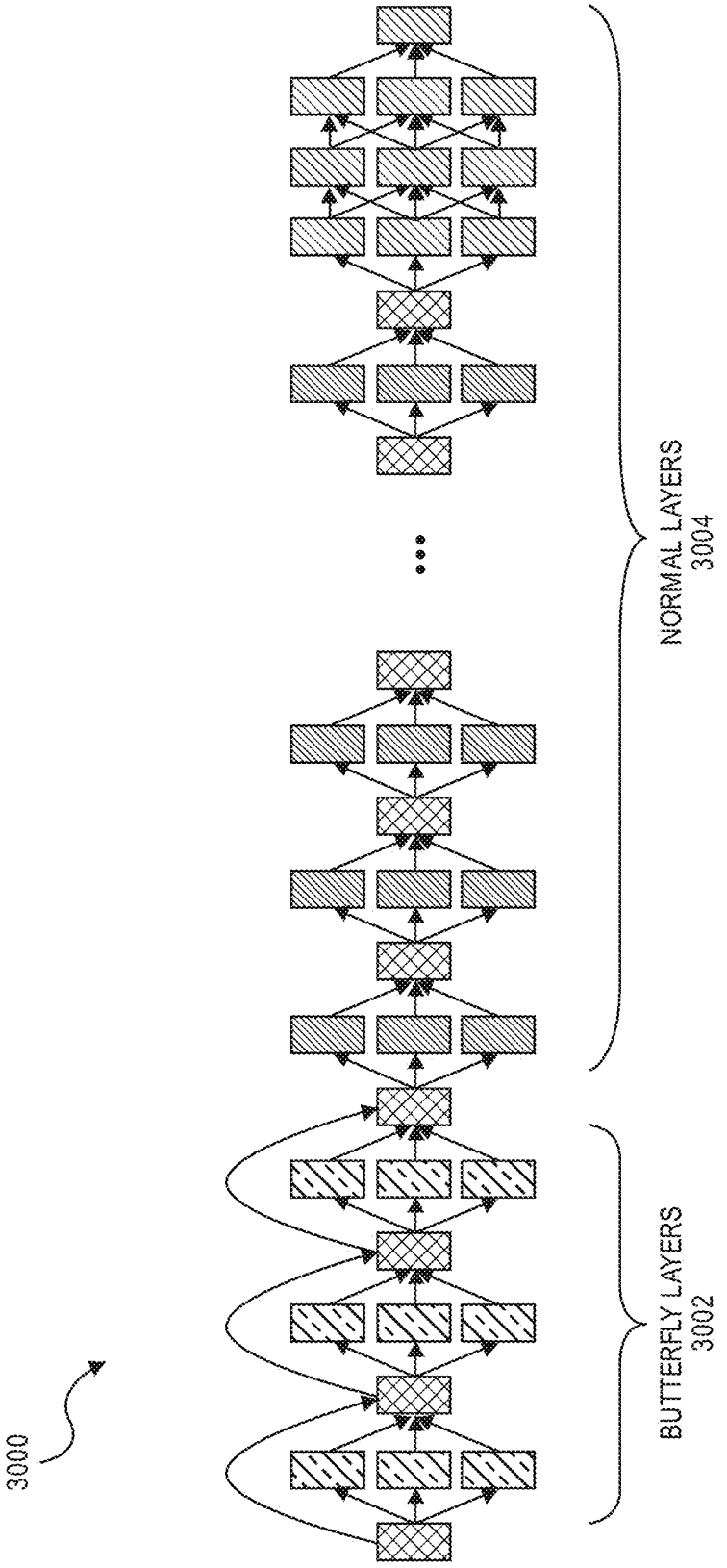


FIG. 30

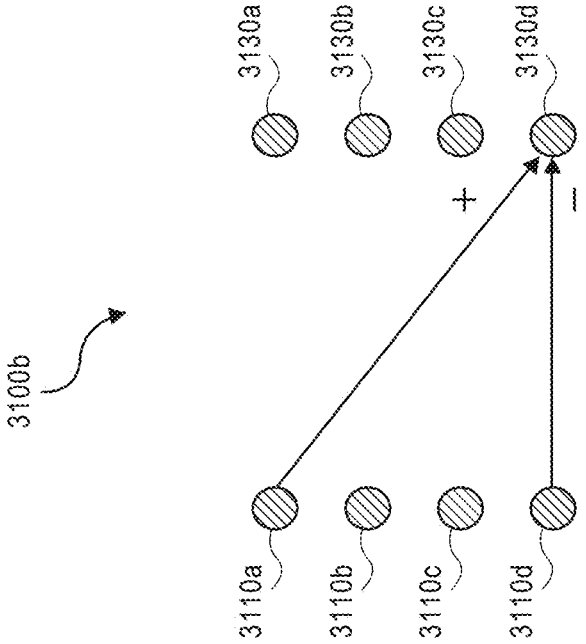


FIG. 31B

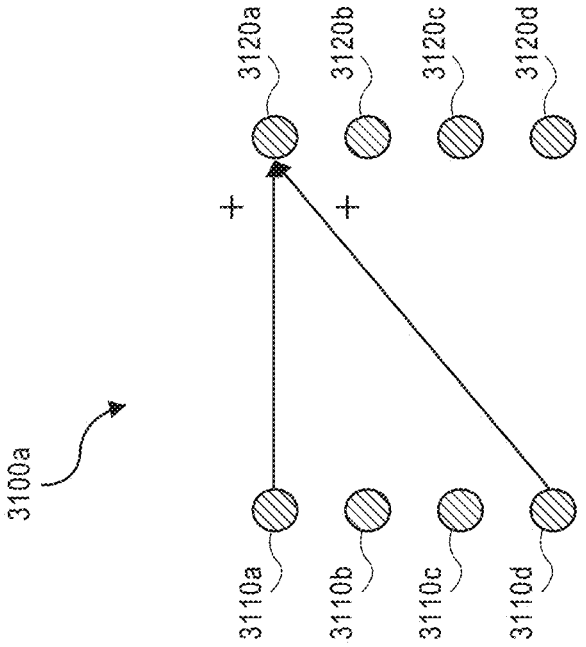


FIG. 31A

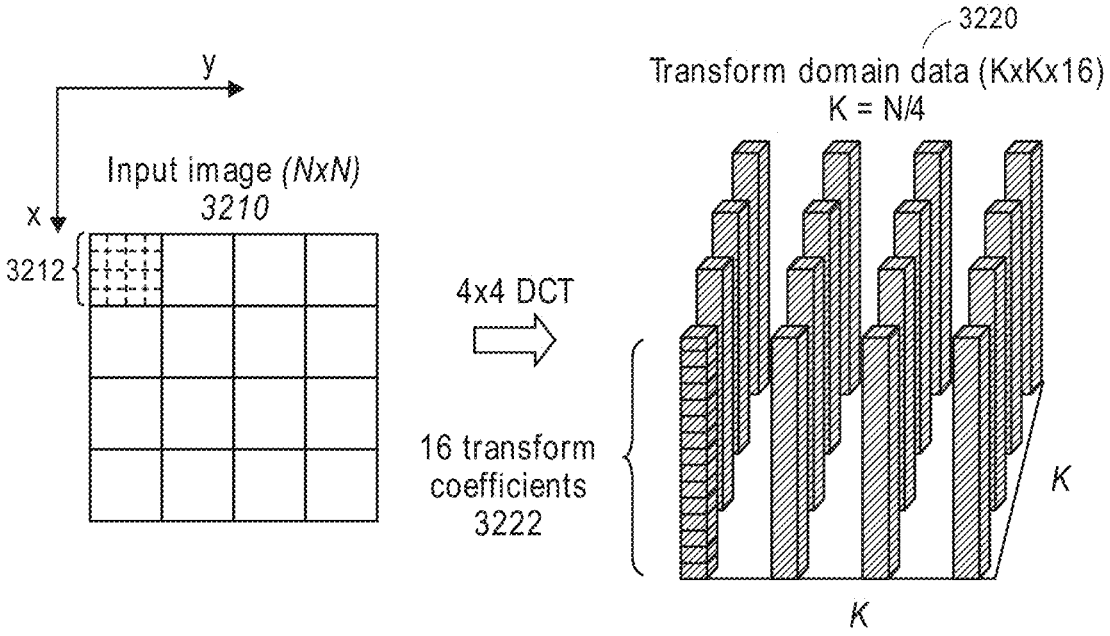

y
x
Input image (NxN)
3210
3212
4x4 DCT
16 transform coefficients
3222
3220
Transform domain data (KxKx16)
K = N/4
K
K

FIG. 32

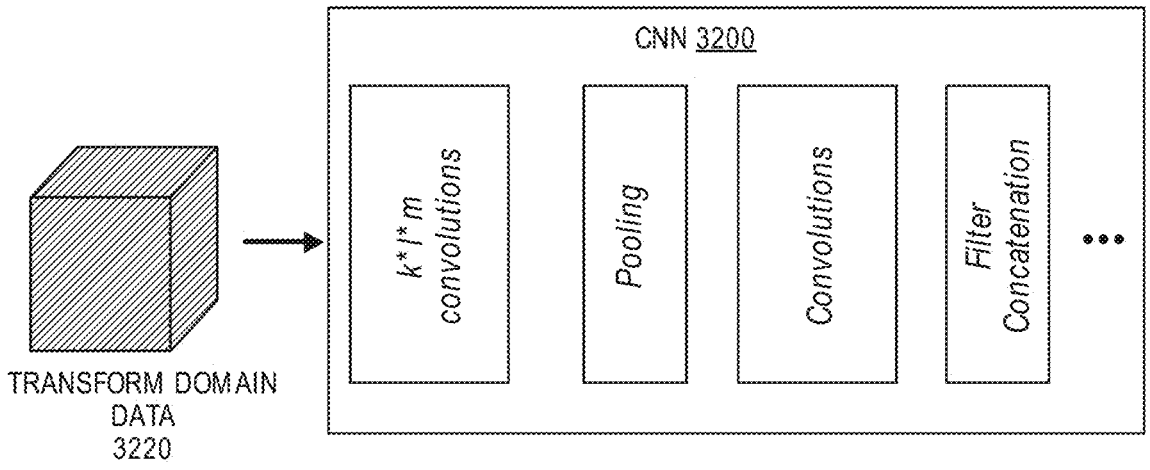

CNN 3200
k*l*m convolutions
Pooling
Convolutions
Filter Concatenation
...
TRANSFORM DOMAIN DATA
3220

FIG. 33

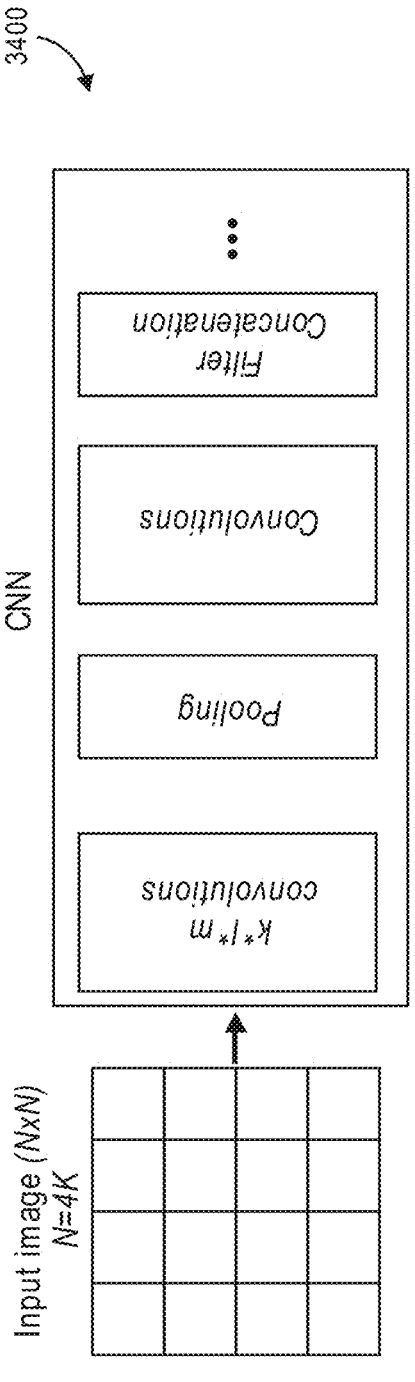

3400
CNN
k*l*m convolutions
Pooling
Convolutions
Filter Concatenation
Input image (NxN) N=4K

FIG. 34

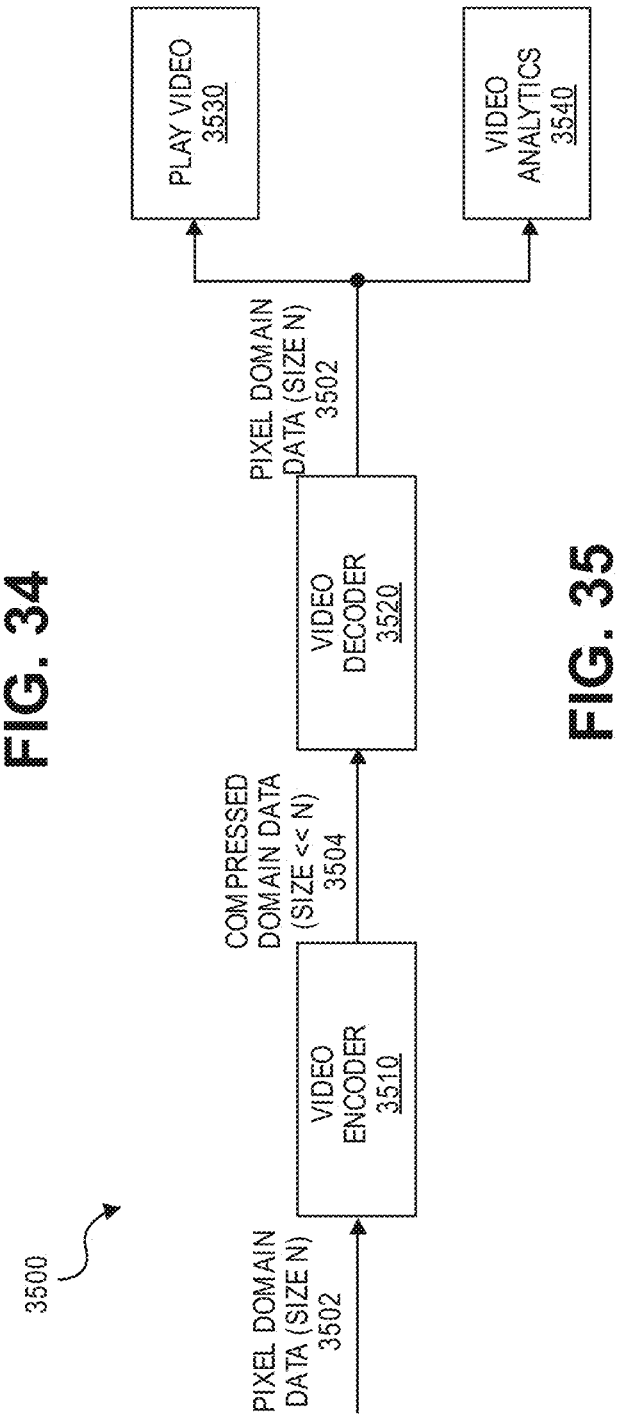

3500
PIXEL DOMAIN DATA (SIZE N) 3502
VIDEO ENCODER 3510
COMPRESSED DOMAIN DATA (SIZE << N) 3504
VIDEO DECODER 3520
PIXEL DOMAIN DATA (SIZE N) 3502
PLAY VIDEO 3530
VIDEO ANALYTICS 3540

FIG. 35

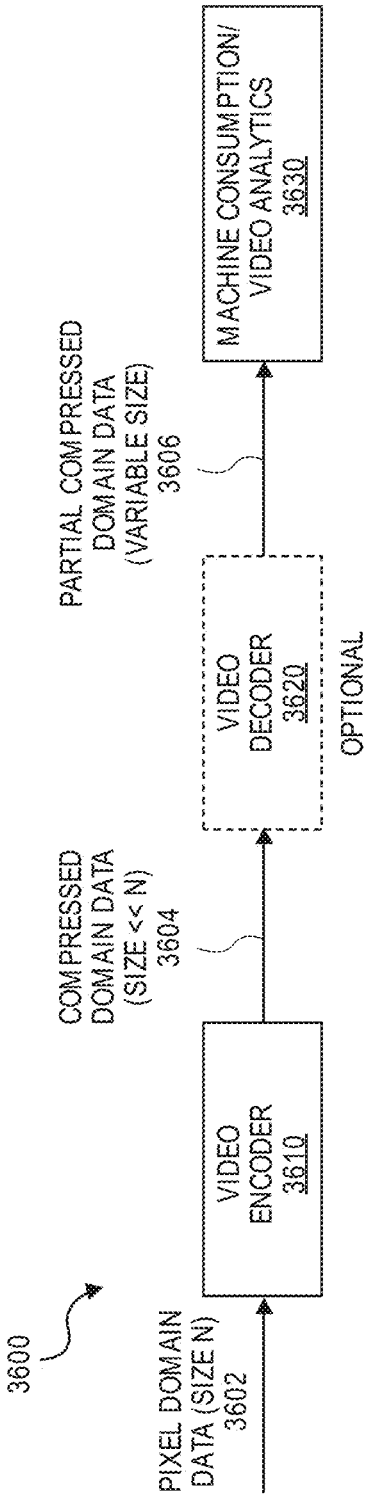

3600
PIXEL DOMAIN DATA (SIZE N) 3602
VIDEO ENCODER 3610
COMPRESSED DOMAIN DATA (SIZE << N) 3604
VIDEO DECODER 3620
OPTIONAL
PARTIAL COMPRESSED DOMAIN DATA (VARIABLE SIZE) 3606
MACHINE CONSUMPTION/ VIDEO ANALYTICS 3630

FIG. 36

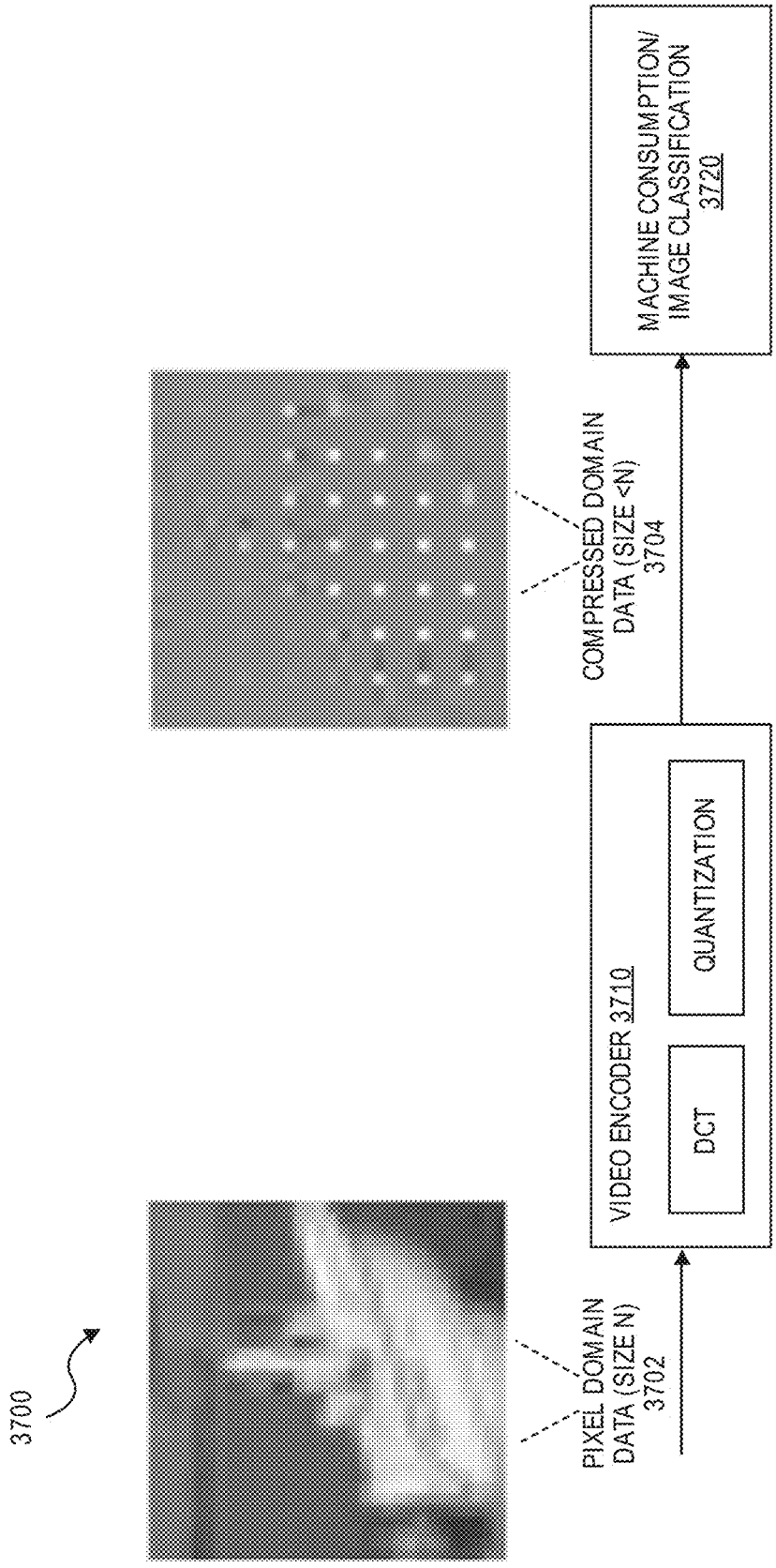

3700
PIXEL DOMAIN DATA (SIZE N) 3702
VIDEO ENCODER 3710
DCT
QUANTIZATION
COMPRESSED DOMAIN DATA (SIZE <N) 3704
MACHINE CONSUMPTION/ IMAGE CLASSIFICATION 3720

FIG. 37

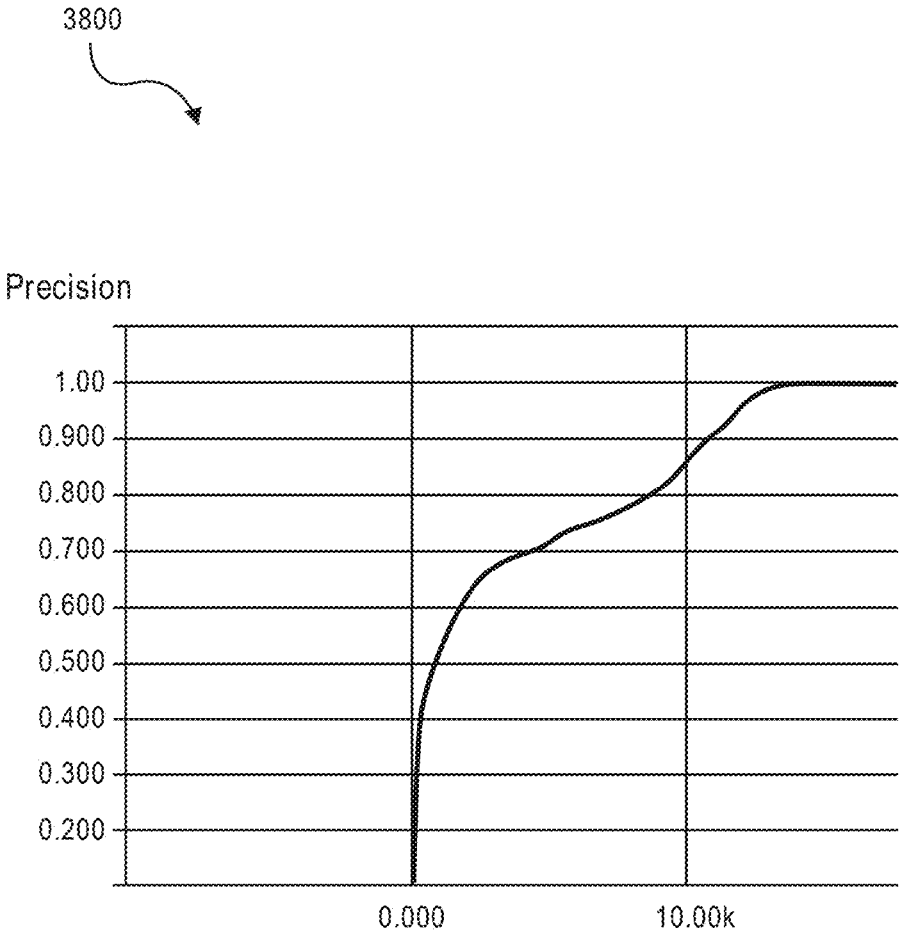

3800
Precision
1.00
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.000
10.00k

FIG. 38

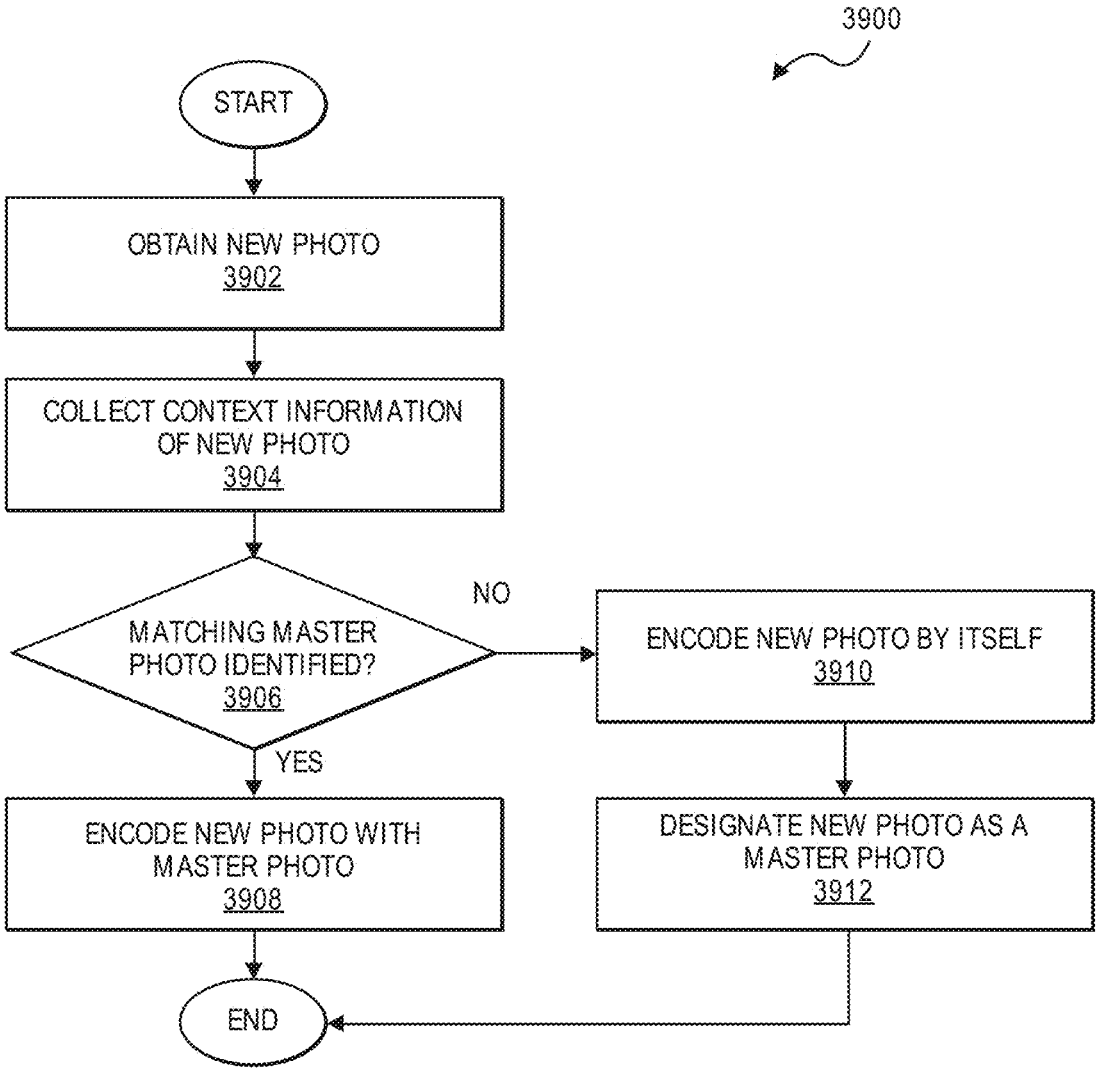

3900
START
OBTAIN NEW PHOTO
3902
COLLECT CONTEXT INFORMATION OF NEW PHOTO
3904
MATCHING MASTER PHOTO IDENTIFIED?
3906
NO
YES
ENCODE NEW PHOTO BY ITSELF
3910
ENCODE NEW PHOTO WITH MASTER PHOTO
3908
DESIGNATE NEW PHOTO AS A MASTER PHOTO
3912
END

FIG. 39

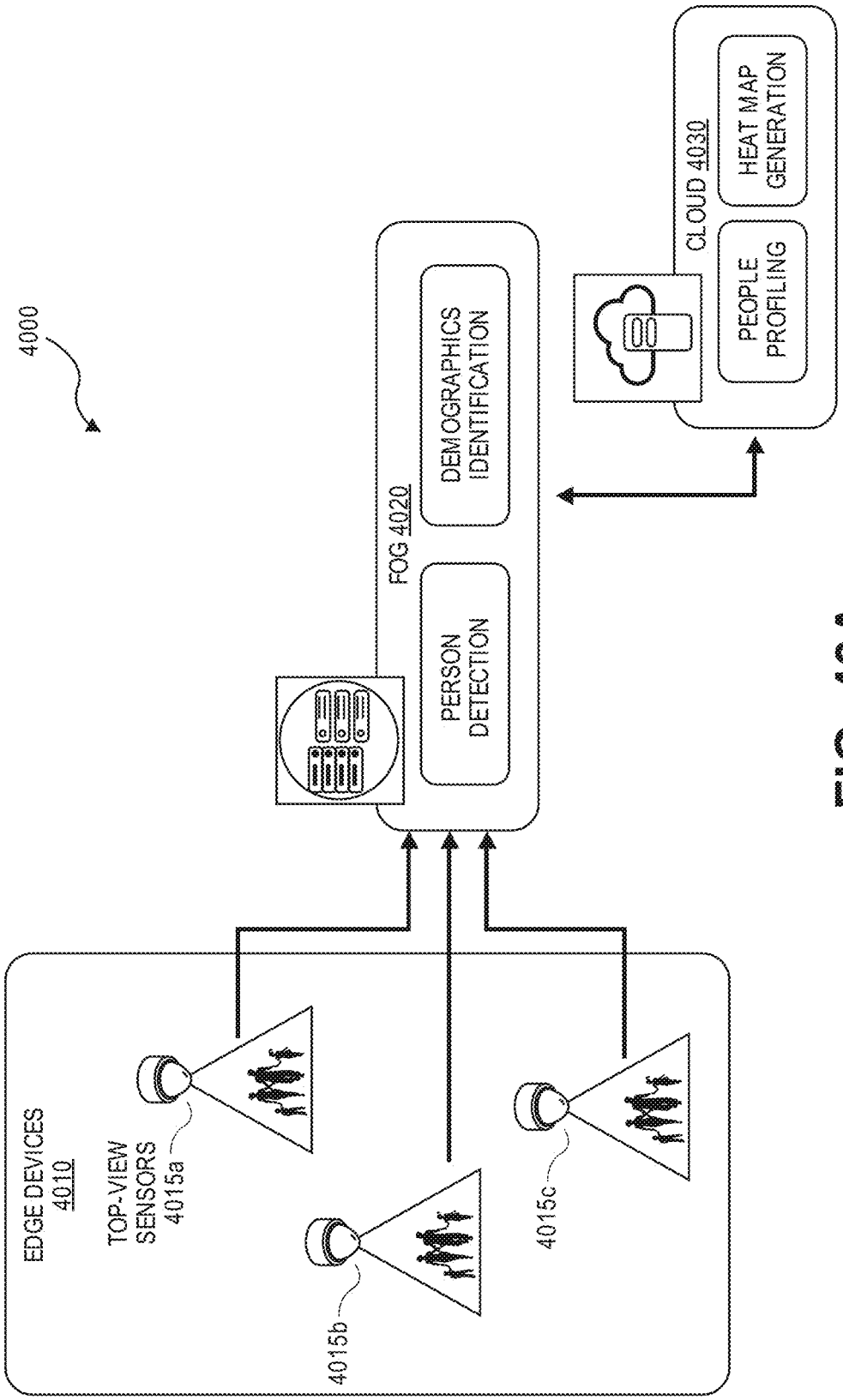

4000
EDGE DEVICES
4010
TOP-VIEW
SENSORS
4015a
4015b
4015c
FOG 4020
PERSON
DETECTION
DEMOGRAPHICS
IDENTIFICATION
CLOUD 4030
PEOPLE
PROFILING
HEAT MAP
GENERATION

FIG. 40A

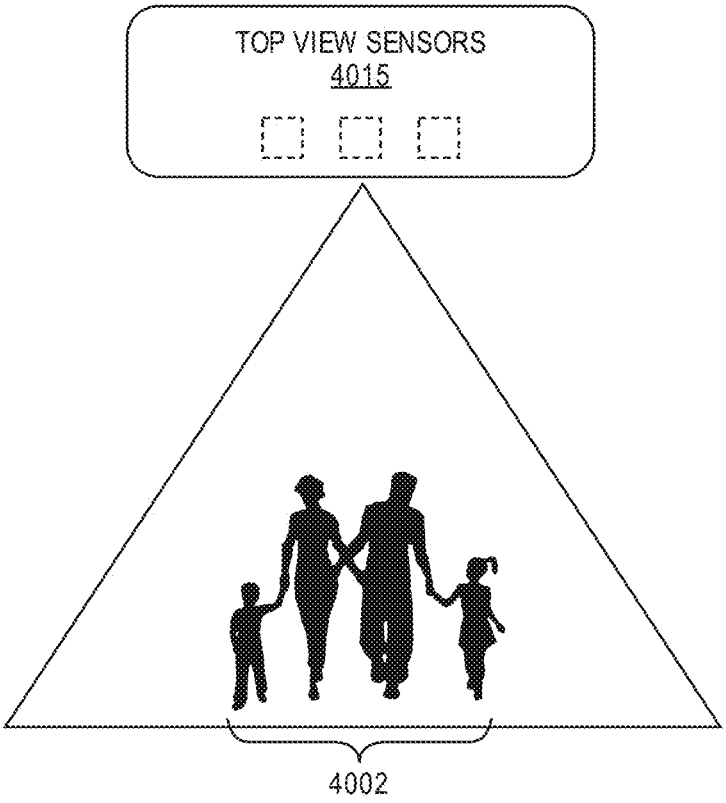


FIG. 40B

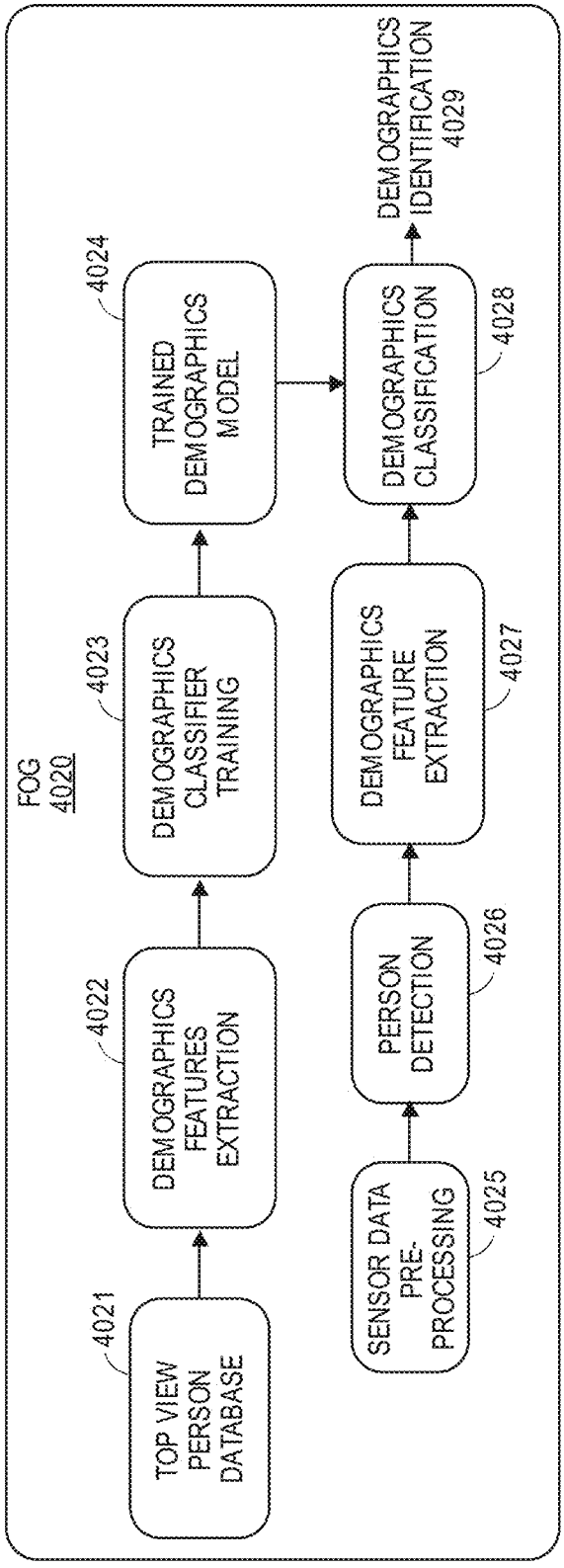

FOG
4020
TOP VIEW PERSON DATABASE
4021
DEMOGRAPHICS FEATURES EXTRACTION
4022
DEMOGRAPHICS CLASSIFIER TRAINING
4023
TRAINED DEMOGRAPHICS MODEL
4024
SENSOR DATA PRE-PROCESSING
4025
PERSON DETECTION
4026
DEMOGRAPHICS FEATURE EXTRACTION
4027
DEMOGRAPHICS CLASSIFICATION
4028
DEMOGRAPHICS IDENTIFICATION
4029

FIG. 40C

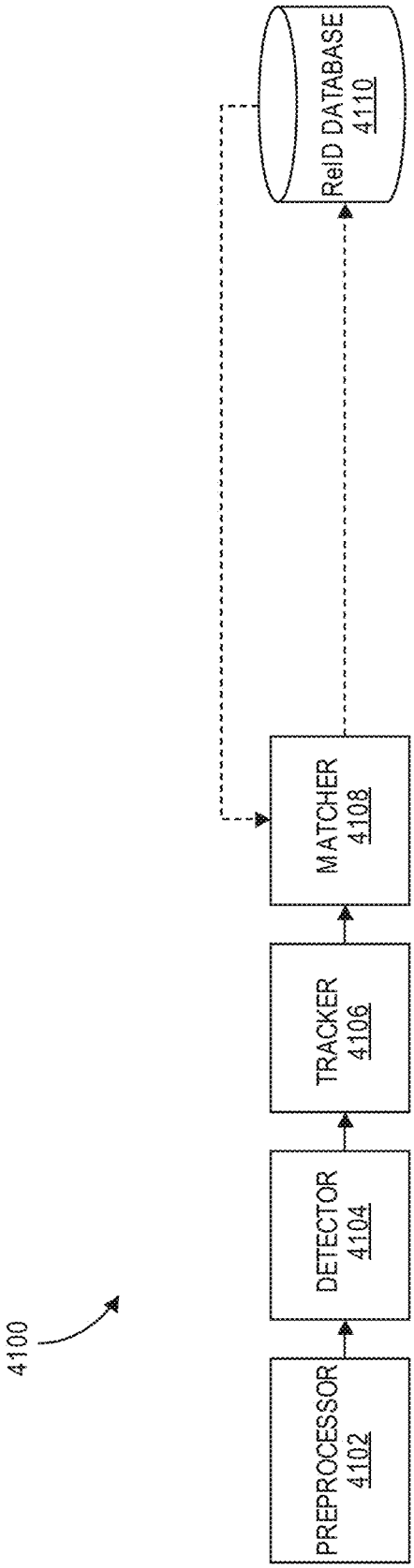

4100
PREPROCESSOR 4102
DETECTOR 4104
TRACKER 4106
MATCHER 4108
ReID DATABASE 4110

FIG. 41

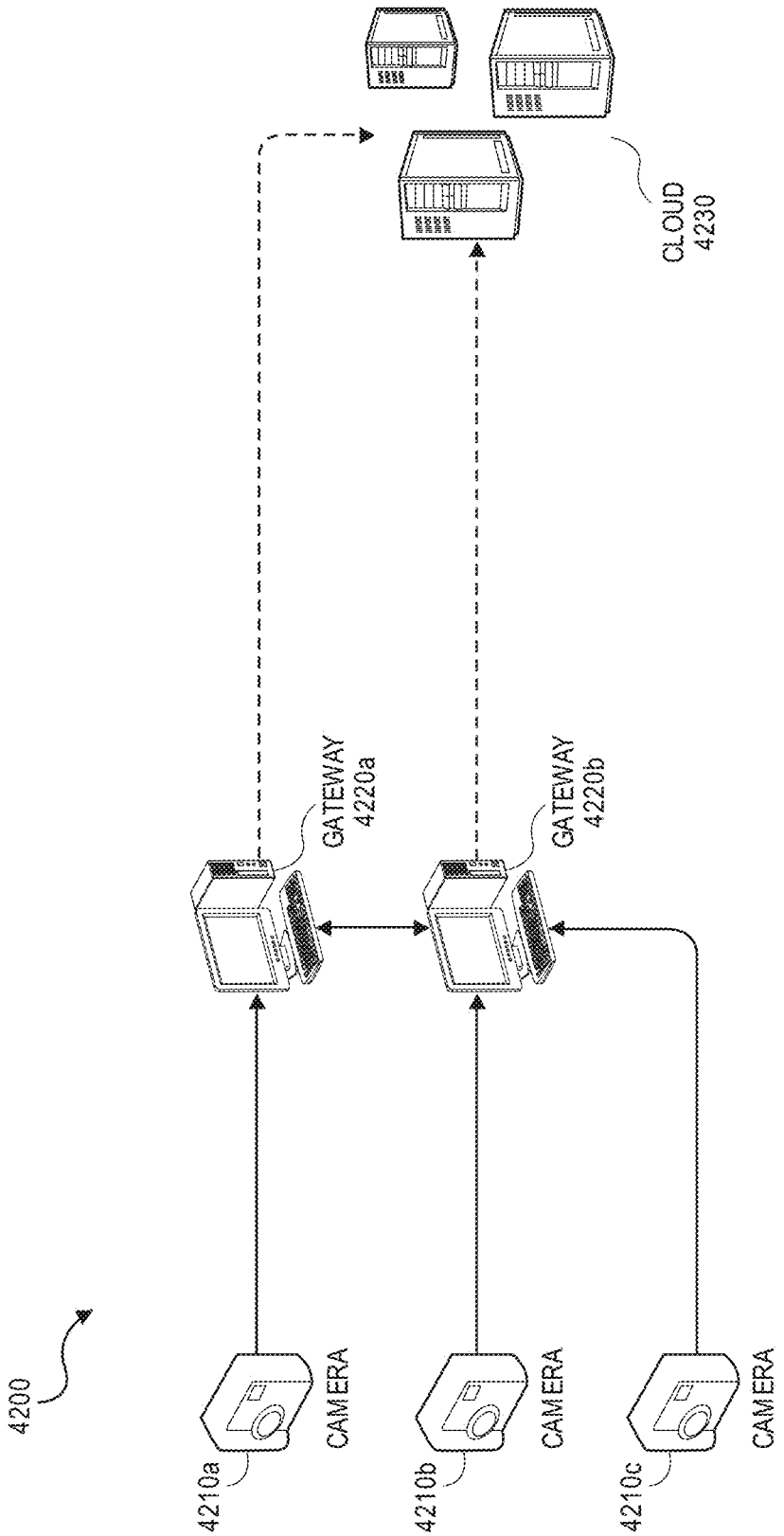

CLOUD
4230
GATEWAY
4220a
GATEWAY
4220b
4200
CAMERA
4210a
CAMERA
4210b
CAMERA
4210c

FIG. 42

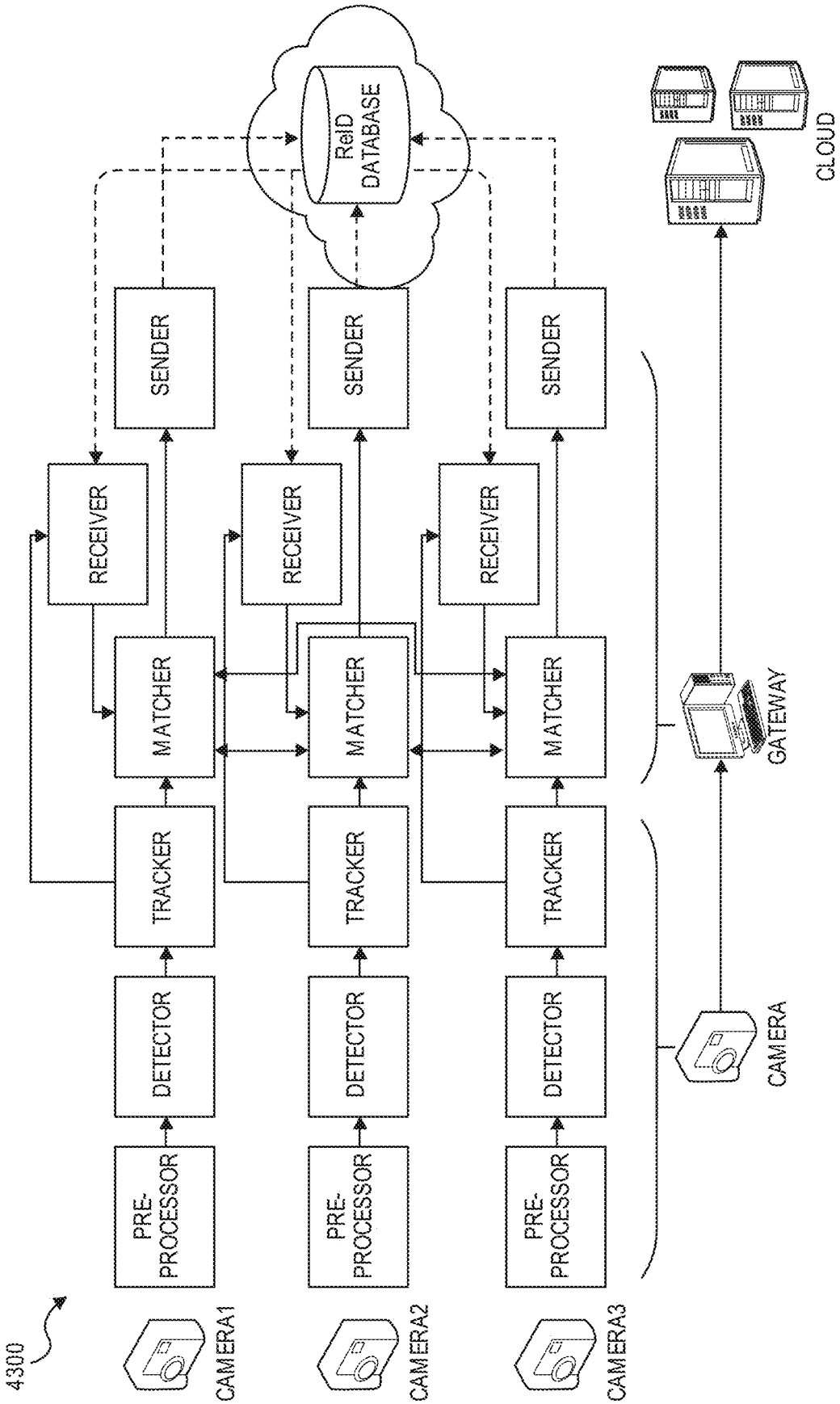

4300
CAMERA1
CAMERA2
CAMERA3
PRE-PROCESSOR
DETECTOR
TRACKER
MATCHER
RECEIVER
SENDER
ReID DATABASE
CLOUD
GATEWAY
CAMERA

FIG. 43

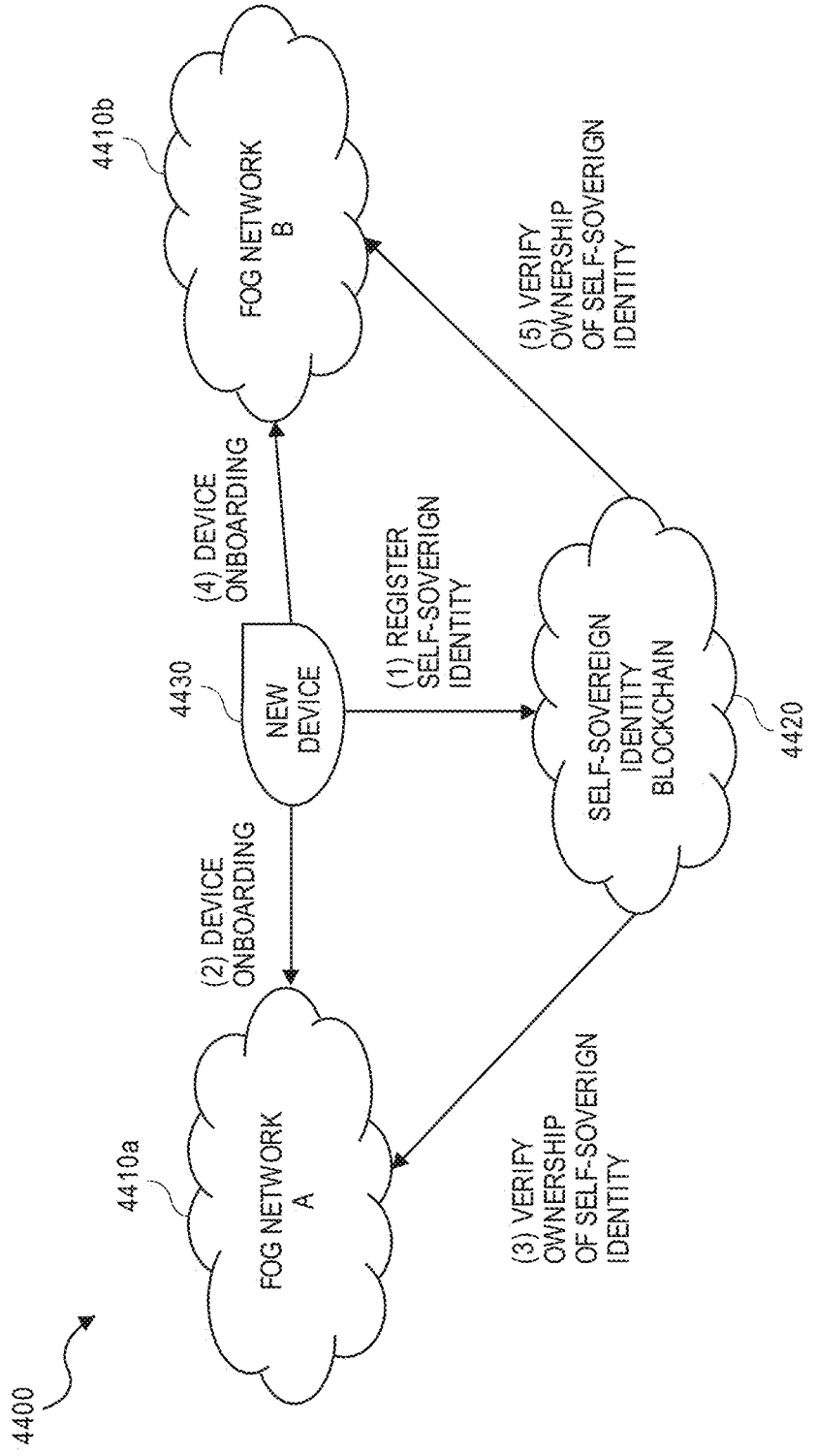

4400
4410a
FOG NETWORK A
4410b
FOG NETWORK B
4420
SELF-SOVEREIGN IDENTITY BLOCKCHAIN
4430
NEW DEVICE
(1) REGISTER SELF-SOVERIGN IDENTITY
(2) DEVICE ONBOARDING
(3) VERIFY OWNERSHIP OF SELF-SOVERIGN IDENTITY
(4) DEVICE ONBOARDING
(5) VERIFY OWNERSHIP OF SELF-SOVERIGN IDENTITY

FIG. 44

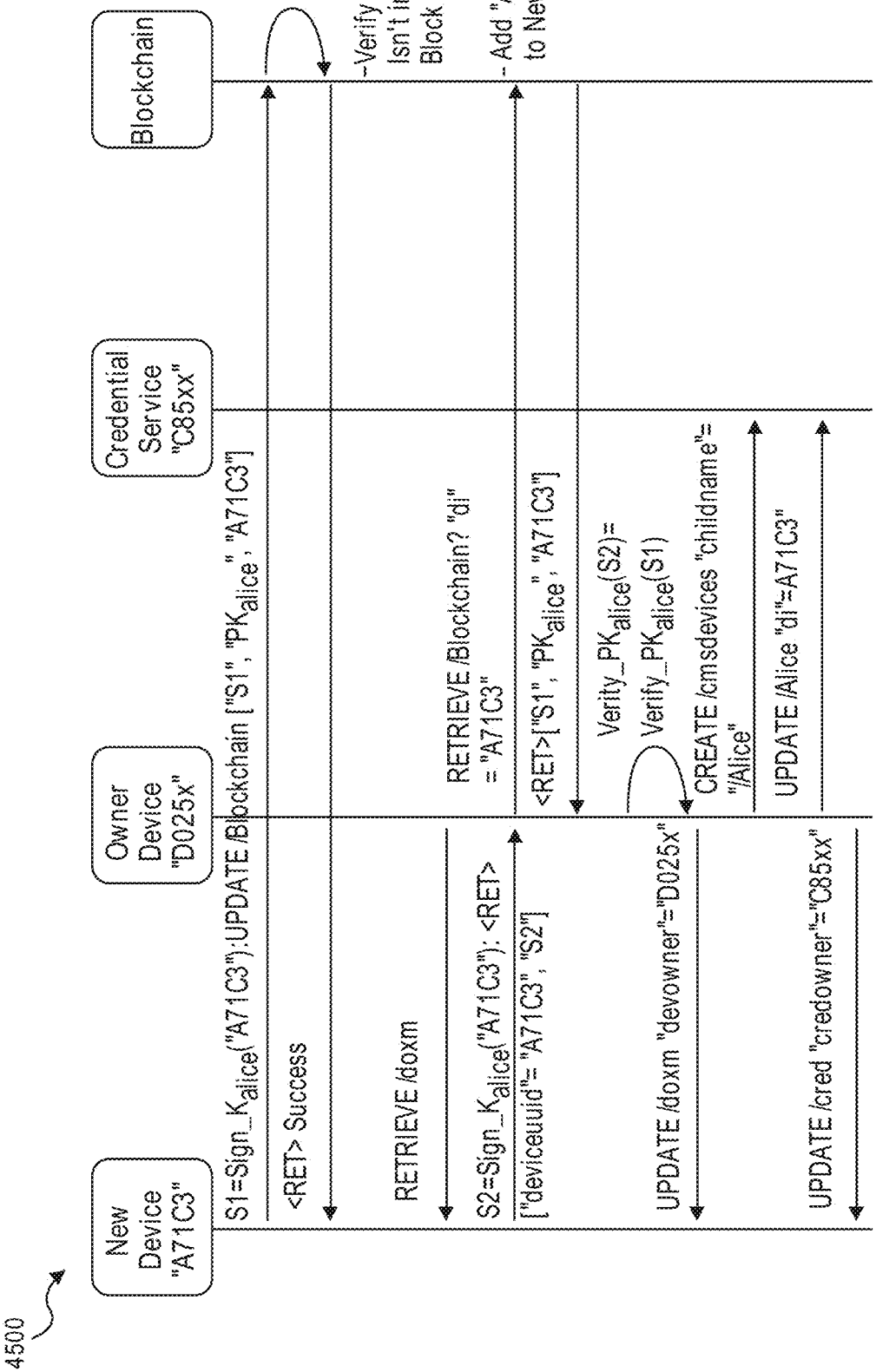

4500
New Device "A71C3"
Owner Device "D025x"
Credential Service "C85xx"
Blockchain
S1=Sign_K_alice("A71C3");UPDATE /Blockchain ["S1", "PK_alice", "A71C3"]
<RET> Success
-Verify "A71C3" Isn't in a prior Block
- Add "A71C3" to New Block
RETRIEVE /doxm
RETRIEVE /Blockchain? "di" = "A71C3"
S2=Sign_K_alice("A71C3"); <RET> ["deviceuuid"= "A71C3", "S2"]
<RET>["S1", "PK_alice", "A71C3"]
Verify_PK_alice(S2)= Verify_PK_alice(S1)
UPDATE /doxm "devowner"="D025x"
CREATE /cmsdevices "childname"= "/Alice"
UPDATE /Alice "di"=A71C3"
UPDATE /cred "credowner"="C85xx"

FIG. 45

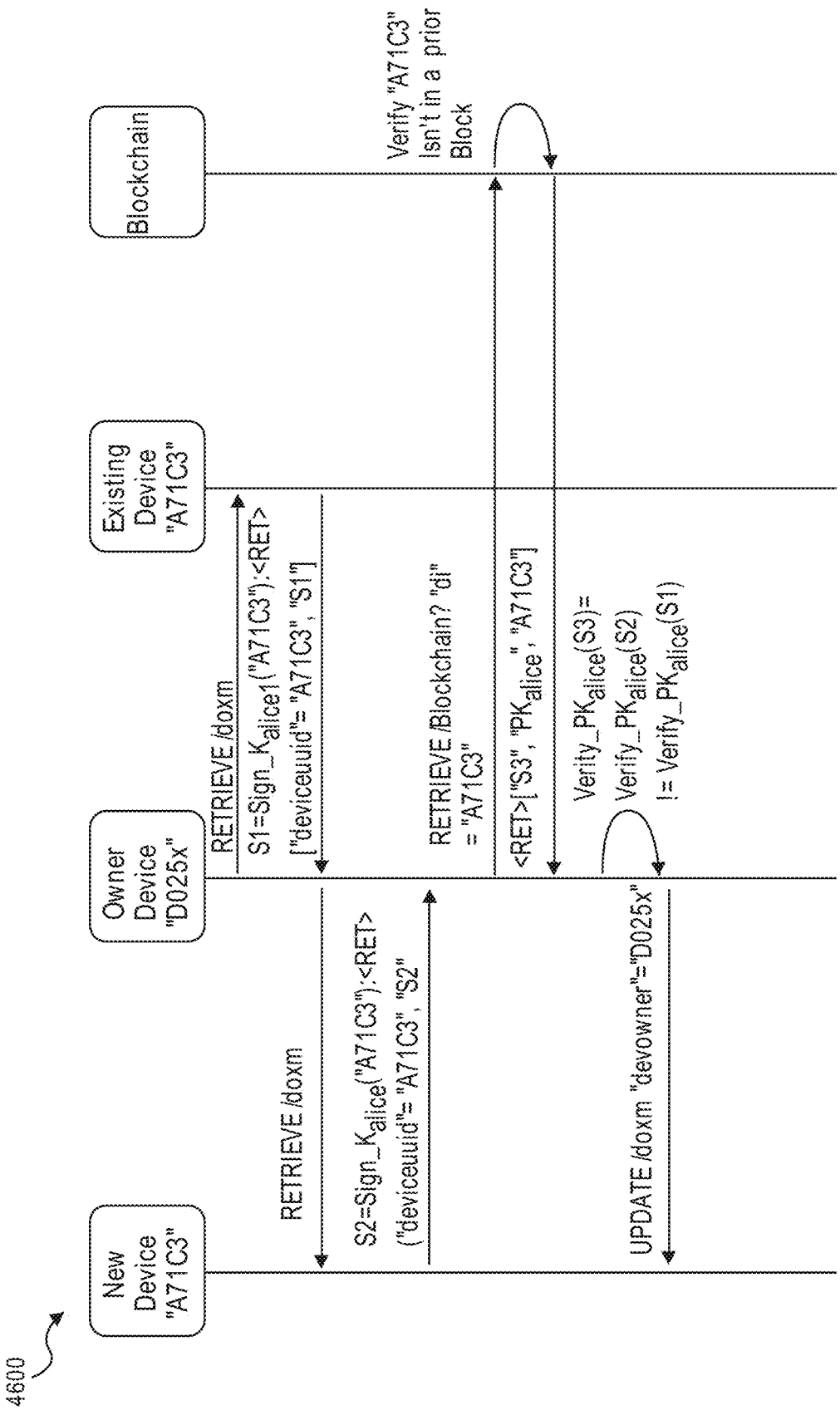

4600
New Device "A71C3"
Owner Device "D025x"
Existing Device "A71C3"
Blockchain
RETRIEVE /doxm
S1=Sign_K_alice1("A71C3");<RET> ["deviceuuid"= "A71C3", "S1"]
RETRIEVE /Blockchain? "di" = "A71C3"
Verify "A71C3" Isn't in a prior Block
<RET>["S3", "PK_alice", "A71C3"]
Verity_PK_alice(S3)= Verify_PK_alice(S2) != Verify_PK_alice(S1)
RETRIEVE /doxm
S2=Sign_K_alice("A71C3");<RET> ("deviceuuid"= "A71C3", "S2"
UPDATE /doxm "devowner"="D025x"

FIG. 46

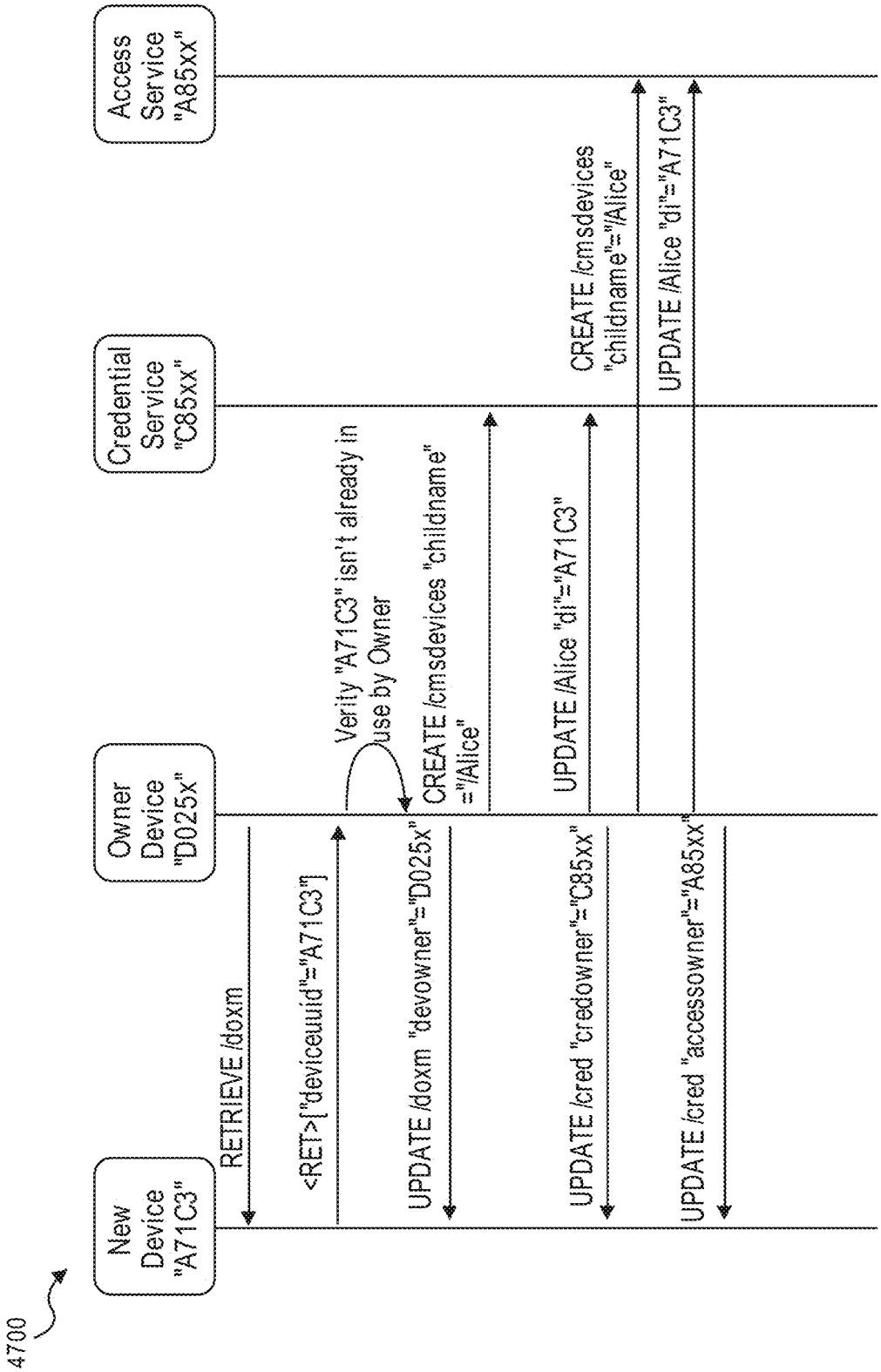

4700
New Device "A71C3"
Owner Device "D025x"
Credential Service "C85xx"
Access Service "A85xx"
RETRIEVE /doxm
<RET>{"deviceuuid"="A71C3"}
Verify "A71C3" isn't already in use by Owner
UPDATE /doxm "devowner"="D025x"
CREATE /cmsdevices "childname" ="/Alice"
UPDATE /cred "credowner"="C85xx"
UPDATE /Alice "di"="A71C3"
CREATE /cmsdevices "childname"="/Alice"
UPDATE /cred "accessowner"="A85xx"
UPDATE /Alice "di"="A71C3"

FIG. 47

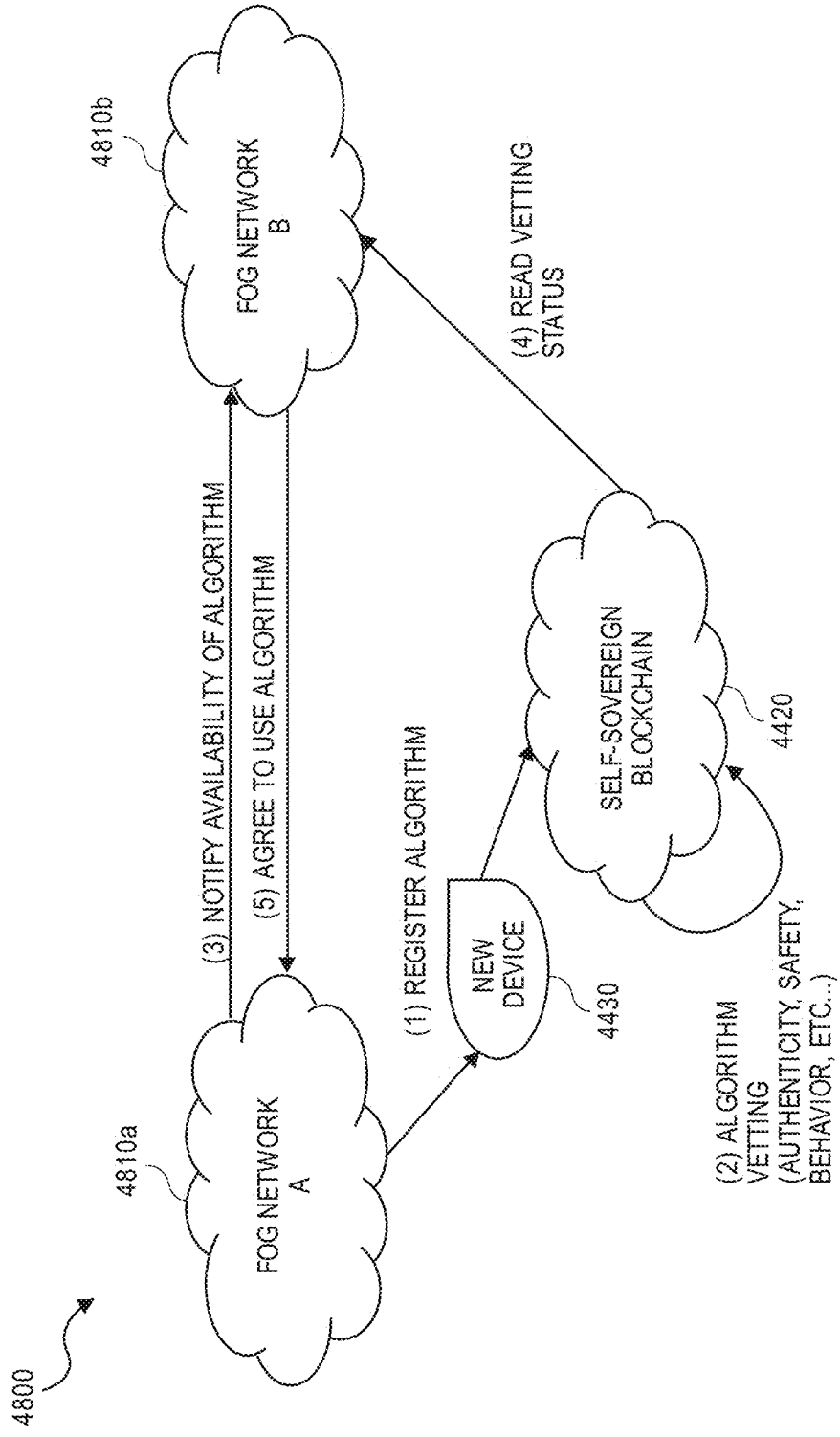

4800
4810a
FOG NETWORK A
4810b
FOG NETWORK B
(1) REGISTER ALGORITHM
NEW DEVICE
4430
(2) ALGORITHM VETTING (AUTHENTICITY, SAFETY, BEHAVIOR, ETC...)
SELF-SOVEREIGN BLOCKCHAIN
4420
(3) NOTIFY AVAILABILITY OF ALGORITHM
(4) READ VETTING STATUS
(5) AGREE TO USE ALGORITHM

FIG. 48

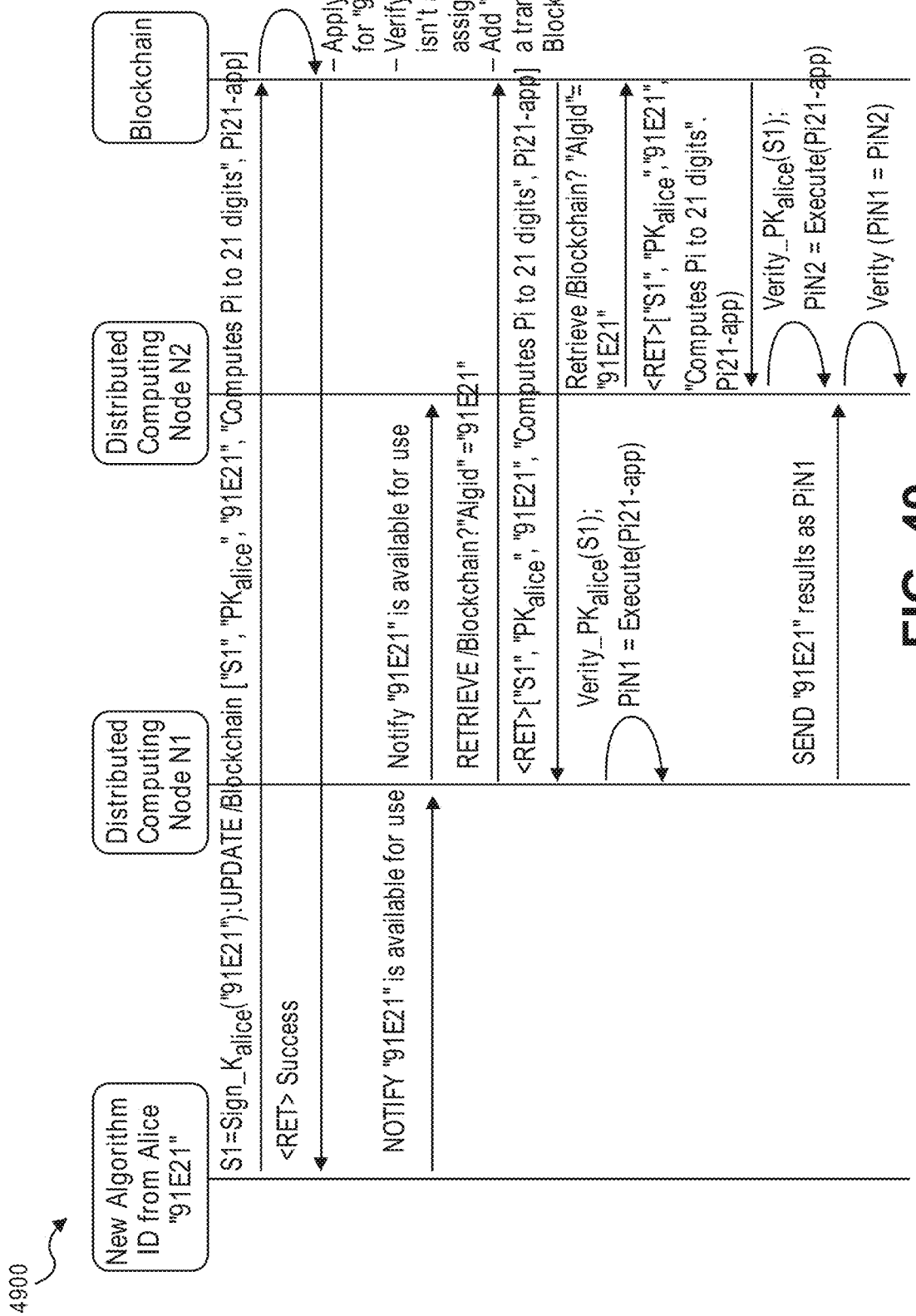

4900
New Algorithm ID from Alice "91E21"
Distributed Computing Node N1
Distributed Computing Node N2
Blockchain
S1=Sign_K_alice("91E21"):UPDATE /Blockchain ["S1", "PK_alice", "91E21", "Computes Pi to 21 digits", Pi21-app]
– Apply vetting for "91E21"
– Verify "91E21" isn't already assigned
– Add "91E21" to a transaction Block
<RET> Success
NOTIFY "91E21" is available for use
Notify "91E21" is available for use
RETRIEVE /Blockchain?"Algid" ="91E21"
<RET>["S1", "PK_alice", "91E21", "Computes Pi to 21 digits", Pi21-app]
Retrieve /Blockchain? "Algid"= "91E21"
Verify_PK_alice(S1); PiN1 = Execute(Pi21-app)
<RET>["S1", "PK_alice", "91E21", "Computes Pi to 21 digits", Pi21-app)
Verify_PK_alice(S1); PiN2 = Execute(Pi21-app)
SEND "91E21" results as PiN1
Verify (PiN1 = PiN2)

FIG. 49

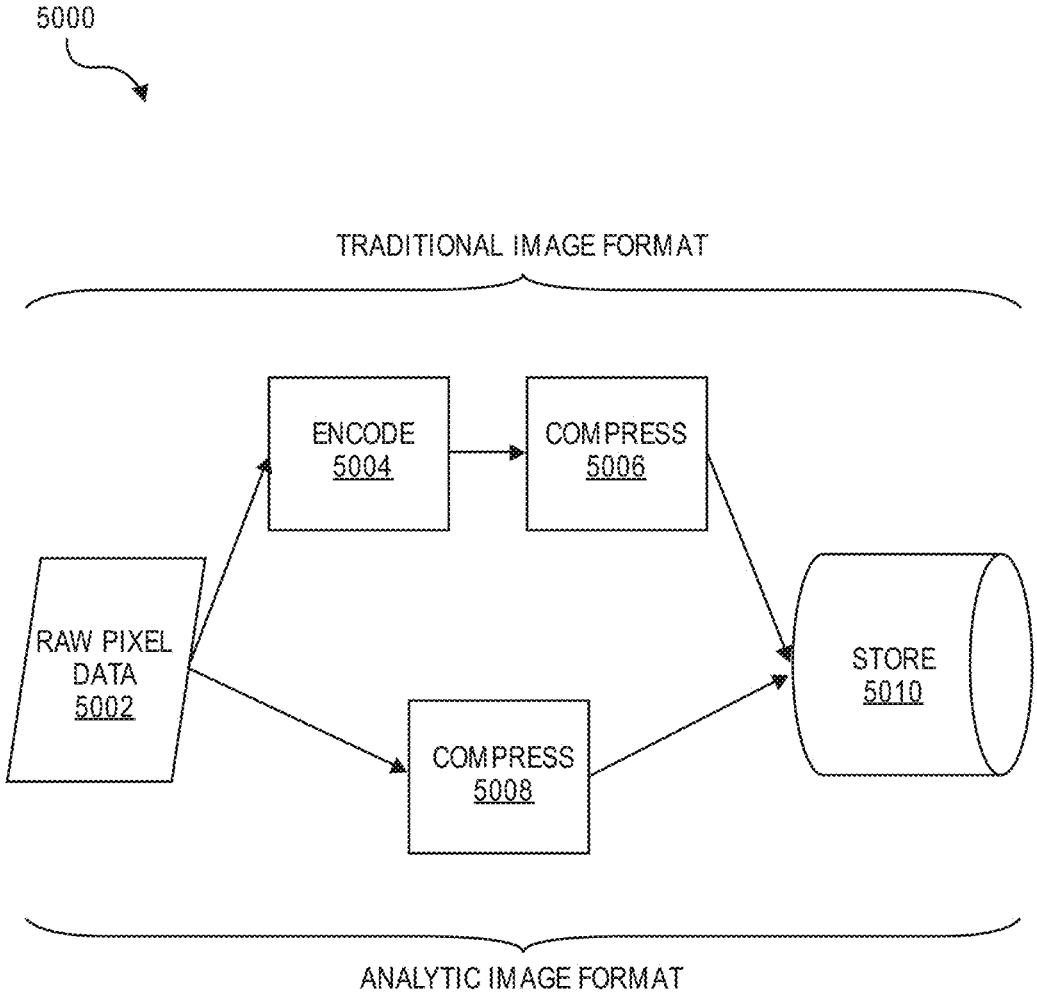

5000
TRADITIONAL IMAGE FORMAT
ENCODE
5004
COMPRESS
5006
RAW PIXEL
DATA
5002
STORE
5010
COMPRESS
5008
ANALYTIC IMAGE FORMAT

FIG. 50

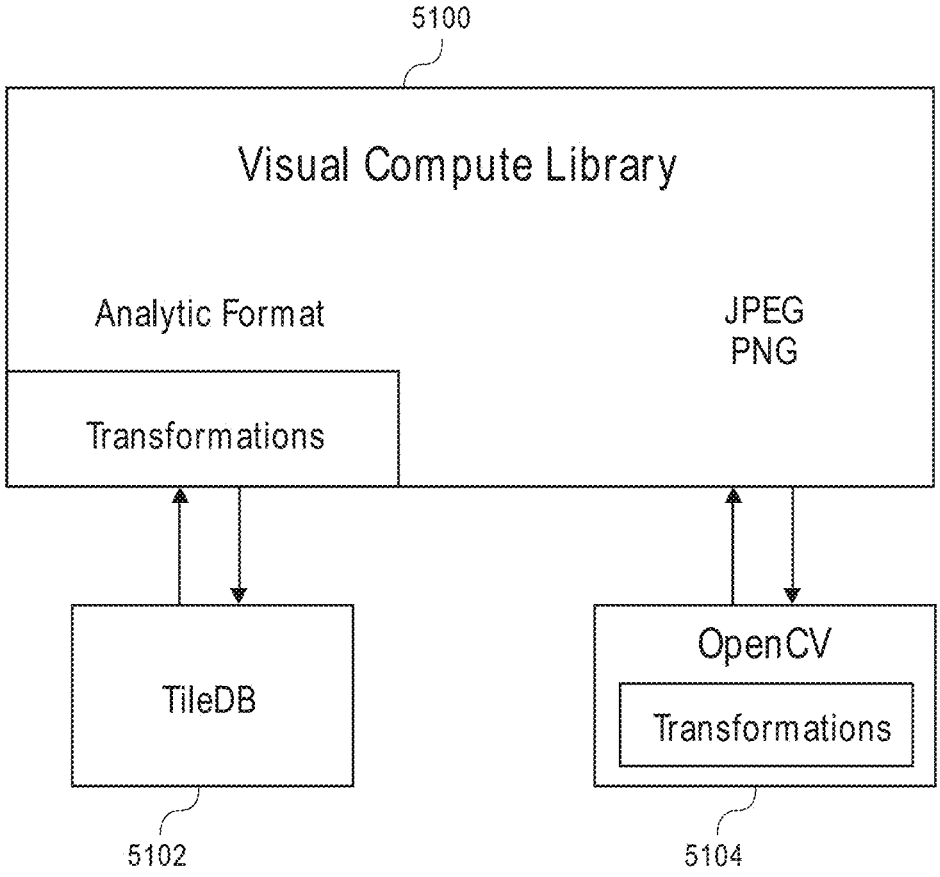

5100
Visual Compute Library
Analytic Format
JPEG
PNG
Transformations
TileDB
OpenCV
Transformations
5102
5104

FIG. 51

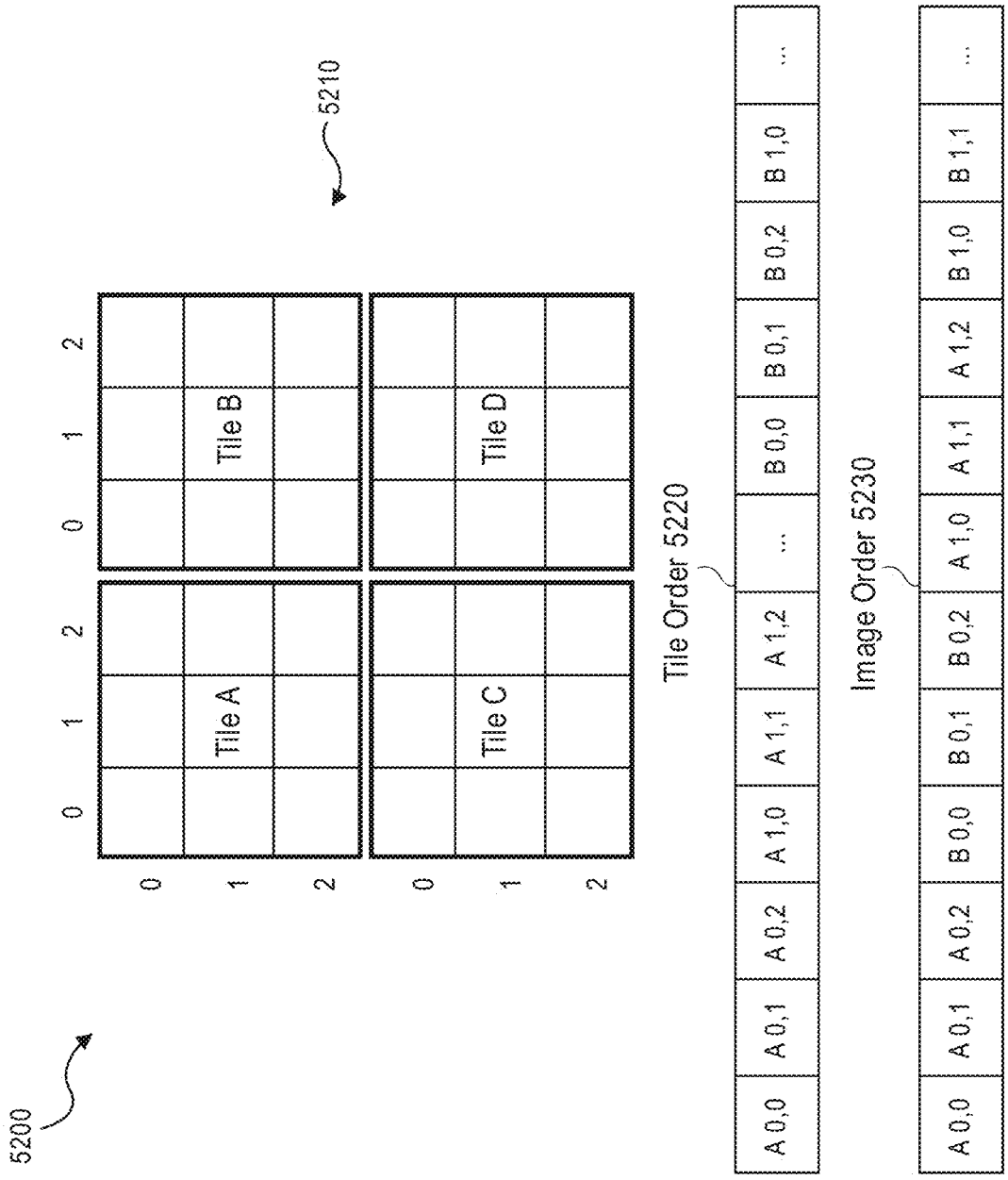

5200
5210
Tile A
Tile B
Tile C
Tile D
0
1
2
Tile Order 5220
A 0,0
A 0,1
A 0,2
A 1,0
A 1,1
A 1,2
...
B 0,0
B 0,1
B 0,2
B 1,0
...
Image Order 5230
A 0,0
A 0,1
A 0,2
B 0,0
B 0,1
B 0,2
A 1,0
A 1,1
A 1,2
B 1,0
B 1,1
...

FIG. 52

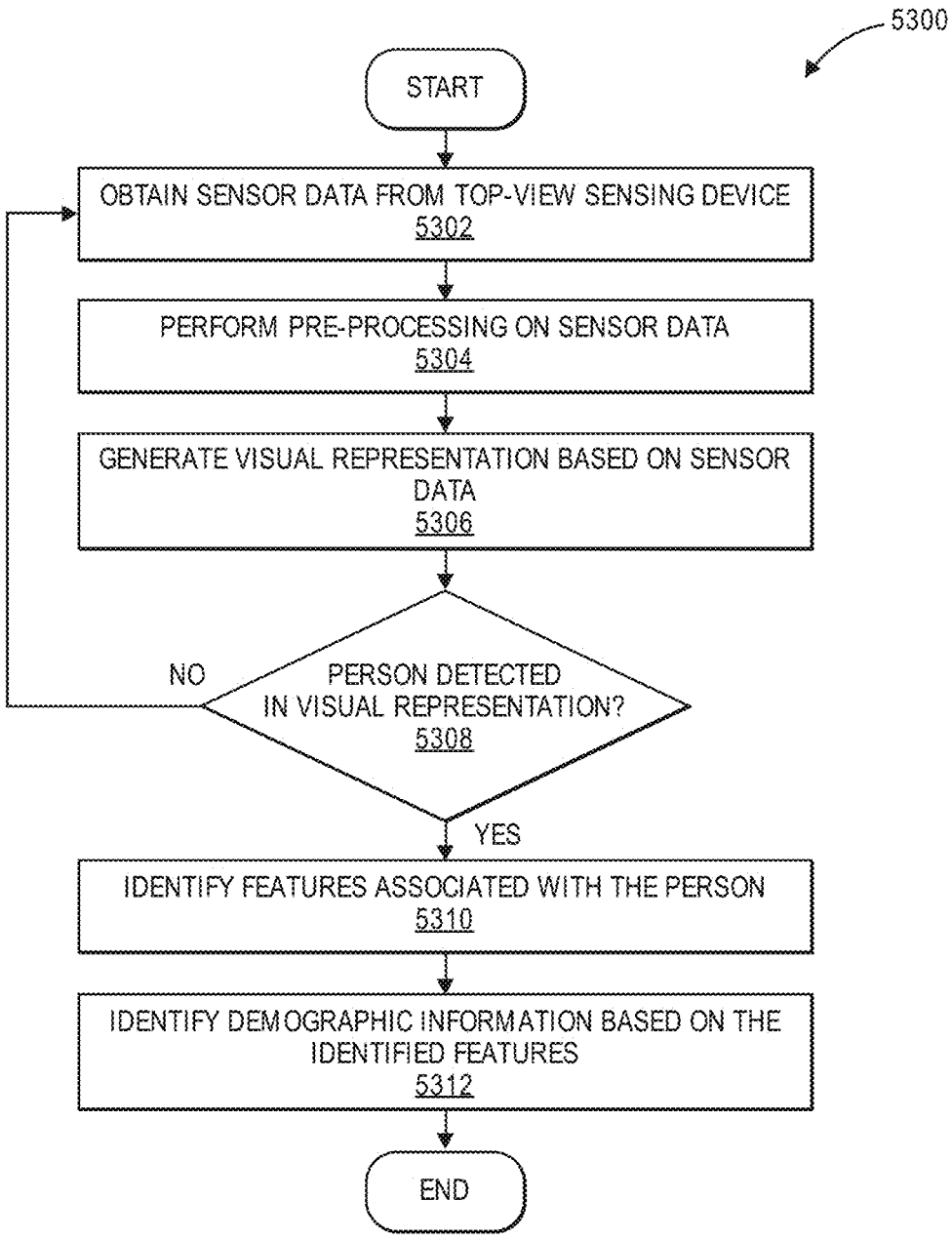

5300
START
OBTAIN SENSOR DATA FROM TOP-VIEW SENSING DEVICE
5302
PERFORM PRE-PROCESSING ON SENSOR DATA
5304
GENERATE VISUAL REPRESENTATION BASED ON SENSOR DATA
5306
PERSON DETECTED IN VISUAL REPRESENTATION?
5308
NO
YES
IDENTIFY FEATURES ASSOCIATED WITH THE PERSON
5310
IDENTIFY DEMOGRAPHIC INFORMATION BASED ON THE IDENTIFIED FEATURES
5312
END

FIG. 53

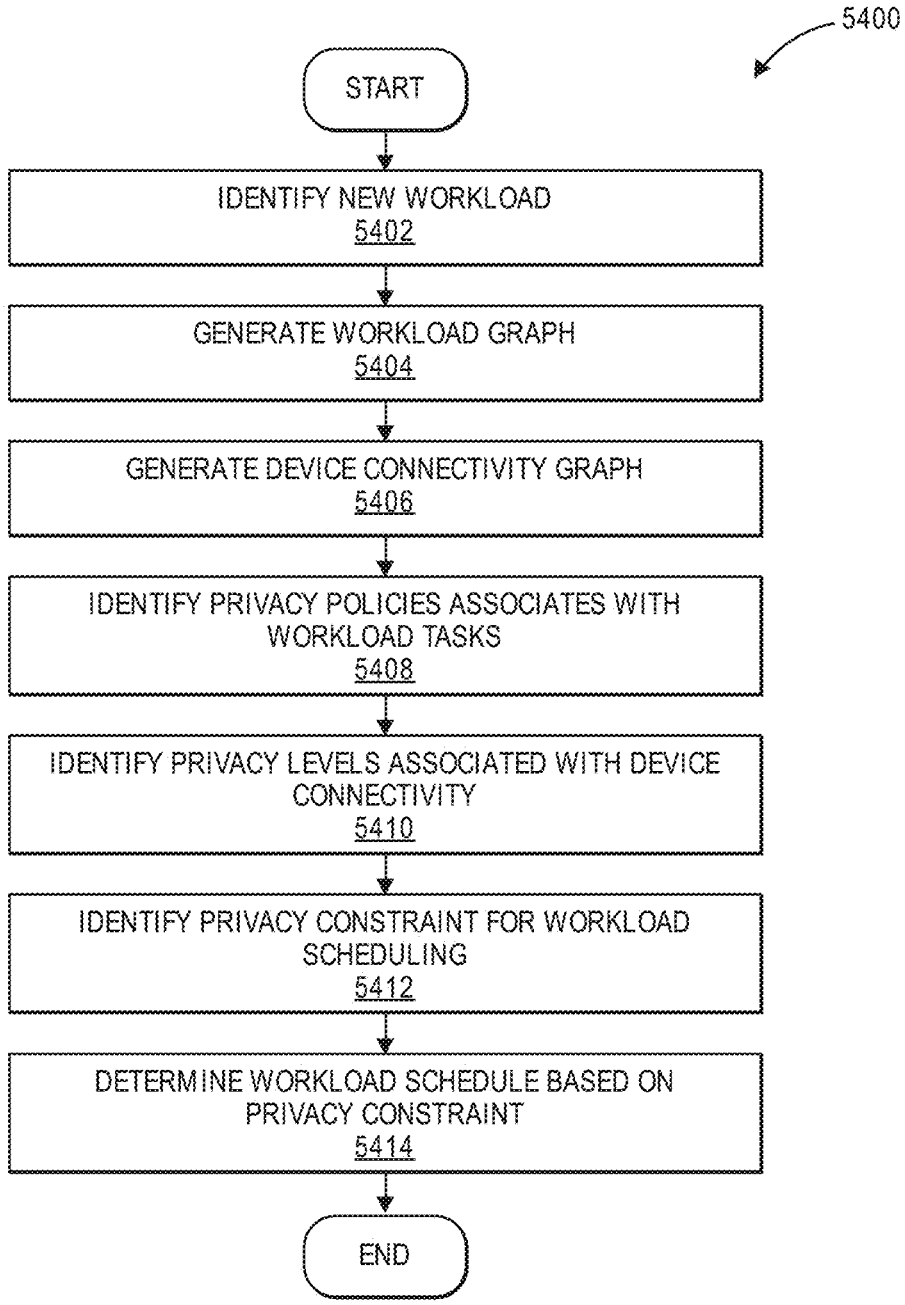

5400
START
IDENTIFY NEW WORKLOAD
5402
GENERATE WORKLOAD GRAPH
5404
GENERATE DEVICE CONNECTIVITY GRAPH
5406
IDENTIFY PRIVACY POLICIES ASSOCIATES WITH WORKLOAD TASKS
5408
IDENTIFY PRIVACY LEVELS ASSOCIATED WITH DEVICE CONNECTIVITY
5410
IDENTIFY PRIVACY CONSTRAINT FOR WORKLOAD SCHEDULING
5412
DETERMINE WORKLOAD SCHEDULE BASED ON PRIVACY CONSTRAINT
5414
END

FIG. 54

PRIVACY-PRESERVING DISTRIBUTED VISUAL DATA PROCESSING

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This patent application is a continuation (and claims the benefit under 35 U.S.C. § 120) of U.S. application Ser. No. 15/859,324, filed Dec. 29, 2017, which claims the benefit of the filing date of U.S. Provisional Patent Application Ser. No. 62/611,536, filed on Dec. 28, 2017, and entitled "VISUAL FOG," the content of which is hereby expressly incorporated by reference.

FIELD OF THE SPECIFICATION

[0002] This disclosure relates in general to the field of computing systems, and more particularly, though not exclusively, to visual computing.

BACKGROUND

[0003] Advancements in modern computing have led to an increased use of visual computing for a variety of mainstream computing applications. In particular, rapid deployments of cameras have been leveraged for numerous visual computing applications that rely on large-scale video analytics and visual data processing. Existing approaches to large-scale visual computing, however, suffer from numerous limitations. For example, existing visual computing approaches are implemented using rigid designs that utilize resources inefficiently and provide limited functionality, privacy, and security. As a result, existing approaches often suffer from high latency and are inaccurate, unreliable, inflexible, and incapable of scaling efficiently.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] The present disclosure is best understood from the following detailed description when read with the accompanying figures. It is emphasized that, in accordance with the standard practice in the industry, various features are not necessarily drawn to scale, and are used for illustration purposes only. Where a scale is shown, explicitly or implicitly, it provides only one illustrative example. In other embodiments, the dimensions of the various features may be arbitrarily increased or reduced for clarity of discussion.

[0005] FIG. **1** illustrates an example embodiment of a visual fog system in accordance with certain embodiments.

[0006] FIGS. **2**, **3**, **4**, and **5** illustrate examples of Internet-of-Things (IoT) networks and architectures that can be used in accordance with certain embodiments.

[0007] FIGS. **6** and **7** illustrate example computer architectures that can be used in accordance with certain embodiments.

[0008] FIG. **8** illustrates an example embodiment of an architecture for visual fog nodes.

[0009] FIGS. **9**, **10**, **11**, and **12**A-B illustrate example embodiments of a visual fog architecture.

[0010] FIGS. **13** and **14** illustrate example embodiments associated with a visual question answering (VQA) framework.

[0011] FIGS. **15** and **16** illustrate example embodiments of device-centric scheduling for visual fog computing.

[0012] FIG. **17** illustrates an example embodiment of a runtime processing pipeline for a visual fog architecture.

[0013] FIG. **18** illustrates an example embodiment of a visual data storage architecture.

[0014] FIG. **19** illustrates an example of a vision processing pipeline that leverages metadata for searching visual data.

[0015] FIGS. **20** and **21** illustrate examples of representing visual metadata using a property graph.

[0016] FIG. **22** illustrates an example embodiment of an analytic image format designed to aid in visual data processing.

[0017] FIG. **23** illustrates a performance graph for various image formats.

[0018] FIGS. **24**A, **24**B, and **24**C illustrate an example embodiment of a multi-domain cascade convolutional neural network (CNN).

[0019] FIGS. **25**A-B, **26**, **27**, **28**, **29**, **30**, and **31**A-B illustrate the use of butterfly operations for a multi-domain convolutional neural network (CNN).

[0020] FIGS. **32** and **33** illustrate an example embodiment of a three-dimensional (3D) CNN for processing compressed visual data.

[0021] FIG. **34** illustrates an example of a pixel-domain CNN.

[0022] FIG. **35** illustrates an example of a pixel-domain visual analytics pipeline.

[0023] FIGS. **36** and **37** illustrate example embodiments of compressed-domain visual analytics pipelines.

[0024] FIG. **38** illustrates a performance graph showing the precision of a CNN trained using compressed visual data.

[0025] FIG. **39** illustrates a flowchart for an example embodiment of context-aware image compression.

[0026] FIGS. **40**A, **40**B, and **40**C illustrate an example embodiment of a privacy-preserving demographic identification system.

[0027] FIGS. **41**, **42**, and **43** illustrate an example embodiment of privacy-preserving distributed visual data processing.

[0028] FIGS. **44**, **45**, and **46** illustrate example embodiments of self-sovereign device identification for distributed computing networks.

[0029] FIG. **47** illustrates an example of device onboarding/commissioning in a visual fog network without conflict resolution.

[0030] FIGS. **48** and **49** illustrate example embodiments of algorithm identification for distributed computing using a self-sovereign blockchain.

[0031] FIGS. **50**, **51**, and **52** illustrate example embodiments for processing traditional and analytic image formats.

[0032] FIG. **53** illustrates a flowchart for an example embodiment of privacy-preserving demographics identification.

[0033] FIG. **54** illustrates a flowchart for an example embodiment of privacy-preserving distributed visual processing.

EMBODIMENTS OF THE DISCLOSURE

[0034] This patent application claims the benefit of the filing date of U.S. Provisional Patent Application Ser. No. 62/611,536, filed on Dec. 28, 2017, and entitled "VISUAL FOG," the content of which is hereby expressly incorporated by reference.

[0035] The following disclosure provides many different embodiments, or examples, for implementing different fea-

tures of the present disclosure. Specific examples of components and arrangements are described below to simplify the present disclosure. These are, of course, merely examples and are not intended to be limiting. Further, the present disclosure may repeat reference numerals and/or letters in the various examples. This repetition is for the purpose of simplicity and clarity and does not in itself dictate a relationship between the various embodiments and/or configurations discussed. Different embodiments may have different advantages, and no particular advantage is necessarily required of any embodiment.

[0036] Example embodiments that may be used to implement the features and functionality of this disclosure will now be described with more particular reference to the attached FIGURES.

[0037] Visual Fog Introduction

[0038] FIG. **1** illustrates an example embodiment of a visual fog system **100** in accordance with certain embodiments. Advancements in modern computing have led to an increased use of computer vision technologies and large-scale visual computing for a variety of mainstream computing applications. In particular, rapid deployments of cameras and other types of computer vision technologies have been leveraged for a variety of visual computing applications that rely on large-scale video analytics and visual data processing. For example, large-scale visual computing can be leveraged for security and surveillance, transportation (e.g., traffic monitoring, navigation, parking, infrastructure planning, security or amber alerts), retail (e.g., customer analytics), enterprise applications, and so forth.

[0039] Existing approaches to large-scale visual computing, however, suffer from numerous limitations. In particular, existing visual computing approaches are implemented using rigid designs that utilize resources inefficiently (e.g., processing, bandwidth, and storage resources) and provide limited functionality. For example, using existing approaches, visual data is typically captured by devices at the edge of a network and simply funneled to the cloud for processing and storage, thus relying heavily on the cloud infrastructure. Due to the large size of visual data, however, this approach typically consumes significant network bandwidth and requires substantial processing and storage resources in the cloud. As a result, existing approaches often suffer from high latency and inefficient resource utilization, and may also be inaccurate, unreliable, inflexible, and incapable of scaling efficiently.

[0040] Accordingly, this disclosure describes various embodiments of a visual fog computing system **100** for performing large-scale visual computing in an efficient and reliable manner. For example, rather than relying exclusively or primarily on cloud resources **130** for visual computing tasks, visual fog system **100** leverages both cloud **130** and edge **110** resources, which may be collectively referred to as the "fog." In this manner, visual fog system **100** can leverage all available "fog" resources to perform visual computing tasks more efficiently, thus improving resource utilization, latency, accuracy, precision, and reliability. Moreover, as described further throughout this disclosure, visual fog system **100** can be implemented using a flexible design that supports ad-hoc queries and is highly scalable, thus rendering it suitable for many visual computing applications and use cases.

[0041] In the illustrated embodiment of FIG. **1**, visual fog system **100** includes edge resources **110** and a plurality of associated visual sensors **120**, cloud resources **130**, and communication networks **150**, which are respectively discussed further below. Moreover, in various embodiments, these components of visual fog system **100** may be implemented some or all aspects of the visual computing functionality described throughout this disclosure in connection with the remaining FIGURES.

[0042] Edge resources **110** may include any equipment, devices, and/or components deployed or connected near the "edge" of a communication network. In the illustrated embodiment, for example, edge resources **110** include end-user devices **112***a,b* (e.g., desktops, laptops, mobile devices), Internet-of-Things (IoT) devices **114**, and gateways or routers **116**, as described further below. Edge resources **110** may communicate with each other and/or with other remote networks and resources (e.g., cloud resources **130**) through one or more communication networks **150**, such as local area network **150***a* and/or wide area network **150***b*. Moreover, in the illustrated embodiment, edge resources **110** collectively include a plurality of visual sensors **120** (e.g., cameras) for capturing visual representations and data associated with their surroundings. In some embodiments, for example, certain end-user devices **112** and/or IoT devices **114** may include one or more cameras and/or other types of visual sensors **120**. Visual sensors **120** may include any type of visual or optical sensors, such as cameras, ultraviolet (UV) sensors, laser rangefinders (e.g., light detection and ranging (LIDAR)), infrared (IR) sensors, electro-optical/infrared (EO/IR) sensors, and so forth.

[0043] End-user devices **112** may include any device that enables or facilitates interaction with a user in visual fog system **100**, including, for example, desktop computers, laptops, tablets, mobile phones and other mobile devices, and wearable devices (e.g., smart watches, smart glasses, headsets), among other examples.

[0044] IoT devices **114** may include any device capable of communicating and/or participating in an Internet-of-Things (IoT) system or network. IoT systems may refer to new or improved ad-hoc systems and networks composed of a variety of different devices (e.g., IoT devices **114**) interoperating and synergizing for a particular application or use case. Such ad-hoc systems are emerging as more and more products and equipment evolve to become "smart," meaning they are controlled or monitored by computer processors and are capable of communicating with other devices. For example, an IoT device **114** may include a computer processor and/or communication interface to allow interoperation with other components of visual fog system **100**, such as with cloud resources **130** and/or other edge resources **110**. IoT devices **114** may be "greenfield" devices that are developed with IoT capabilities from the ground-up, or "brownfield" devices that are created by integrating IoT capabilities into existing legacy devices that were initially developed without IoT capabilities. For example, in some cases, IoT devices **114** may be built from sensors and communication modules integrated in or attached to "things," such as equipment, toys, tools, vehicles, living things (e.g., plants, animals, humans), and so forth. Alternatively, or additionally, certain IoT devices **114** may rely on intermediary components, such as edge gateways or routers **116**, to communicate with the various components of system **100**.

[0045] IoT devices **114** may include various types of sensors for monitoring, detecting, measuring, and generating sensor data and signals associated with characteristics of

their environment. In some embodiments, for example, certain IoT devices **114** may include visual sensors **120** (e.g., cameras) for capturing visual representations and data associated with their surroundings. IoT devices **114** may also include other types of sensors configured to detect characteristics such as movement, weight, physical contact, temperature, wind, noise, light, position, humidity, radiation, liquid, specific chemical compounds, battery life, wireless signals, computer communications, and bandwidth, among other examples. Sensors can include physical sensors (e.g., physical monitoring components) and virtual sensors (e.g., software-based monitoring components). IoT devices **114** may also include actuators to perform various actions in their respective environments. For example, an actuator may be used to selectively activate certain functionality, such as toggling the power or operation of a security system (e.g., alarm, camera, locks) or household appliance (e.g., audio system, lighting, HVAC appliances, garage doors), among other examples.

[0046] Indeed, this disclosure contemplates use of a potentially limitless universe of IoT devices **114** and associated sensors/actuators. IoT devices **114** may include, for example, any type of equipment and/or devices associated with any type of system **100** and/or industry, including transportation (e.g., automobile, airlines), industrial manufacturing, energy (e.g., power plants), telecommunications (e.g., Internet, cellular, and television service providers), retail, medical (e.g., healthcare, pharmaceutical), and/or food and beverage, among others. In the transportation industry, for example, IoT devices **114** may include equipment and devices associated with aircrafts, automobiles, or vessels, such as navigation systems, autonomous flight or driving systems, traffic monitoring and/or planning systems, parking systems, and/or any internal mechanical or electrical components that are monitored by sensors (e.g., engines). IoT devices **114** may also include equipment, devices, and/or infrastructure associated with industrial manufacturing and production, shipping (e.g., cargo tracking), communications networks (e.g., gateways, routers, servers, cellular towers), server farms, electrical power plants, wind farms, oil and gas pipelines, water treatment and distribution, wastewater collection and treatment, and weather monitoring (e.g., temperature, wind, and humidity sensors), among other examples. IoT devices **114** may also include, for example, any type of "smart" device or system, such as smart entertainment systems (e.g., televisions, audio systems, videogame systems), smart household or office appliances (e.g., heat-ventilation-air-conditioning (HVAC) appliances, refrigerators, washers and dryers, coffee brewers), power control systems (e.g., automatic electricity, light, and HVAC controls), security systems (e.g., alarms, locks, cameras, motion detectors, fingerprint scanners, facial recognition systems), and other home automation systems, among other examples. IoT devices **114** can be statically located, such as mounted on a building, wall, floor, ground, lamppost, sign, water tower, or any other fixed or static structure. IoT devices **114** can also be mobile, such as devices in vehicles or aircrafts, drones, packages (e.g., for tracking cargo), mobile devices, and wearable devices, among other examples. Moreover, any type of edge resource **110** may also be considered as an IoT device **114**, including end-user devices **112** and edge gateways **116**, among other examples.

[0047] Edge gateways and/or routers **116** may be used to facilitate communication to and from edge resources **110**. For example, gateways **116** may provide communication capabilities to existing legacy devices that were initially developed without any such capabilities (e.g., "brownfield" IoT devices **114**). Gateways **116** can also be utilized to extend the geographical reach of edge resources **110** with short-range, proprietary, or otherwise limited communication capabilities, such as IoT devices **114** with Bluetooth or ZigBee communication capabilities. For example, gateways **116** can serve as intermediaries between IoT devices **114** and remote networks or services, by providing a front-haul to the IoT devices **114** using their native communication capabilities (e.g., Bluetooth, ZigBee), and providing a back-haul to other networks **150** and/or cloud resources **130** using another wired or wireless communication medium (e.g., Ethernet, Wi-Fi, cellular). In some embodiments, a gateway **116** may be implemented by a dedicated gateway device, or by a general-purpose device, such as another IoT device **114**, end-user device **112**, or other type of edge resource **110**. In some instances, gateways **116** may also implement certain network management and/or application functionality (e.g., visual computing functionality, IoT application and management functionality), either separately or in conjunction with other components, such as cloud resources **130** and/or other edge resources **110**.

[0048] Cloud resources **130** may include any resources or services that are hosted remotely over a network, which may otherwise be referred to as in the "cloud." In some embodiments, for example, cloud resources **130** may be remotely hosted on servers in a datacenter (e.g., application servers, database servers). Cloud resources **130** may include any resources, services, and/or functionality that can be utilized by or for edge resources **110**, including but not limited to, visual computing applications and services, IoT application and management services, data storage, computational services (e.g., data analytics, searching, diagnostics and fault management), security services (e.g., surveillance, alarms, user authentication), mapping and navigation, geolocation services, network or infrastructure management, payment processing, audio and video streaming, messaging, social networking, news, and weather, among other examples.

[0049] Communication networks **150***a,b* may be used to facilitate communication between components of system **100**. In the illustrated embodiment, for example, edge resources **110** are connected to local area network (LAN) **150***a* in order to facilitate communication with each other and/or other remote networks or resources, such as wide area network (WAN) **150***b* and/or cloud resources **130**. In various embodiments, visual fog system **100** may be implemented using any number or type of communication network(s) **150**, including local area networks, wide area networks, public networks, the Internet, cellular networks, Wi-Fi networks, short-range networks (e.g., Bluetooth or ZigBee), and/or any other wired or wireless communication networks or mediums.

[0050] In general, edge resources **110** (and in particular IoT devices **114**) may generate an extremely large volume and variety of data. As one example, edge resources **110** with visual sensors **120** may generate large volumes of visual data, such as video and/or images. Edge resources **110** typically offload this data to the cloud **130** for processing and/or storage. Cloud resources **130**, however, may not necessarily be suited to handle the rapidly growing volume, variety, and velocity of data generated by IoT devices **114** and other edge resources **110**. For example, cloud-based

processing may not be ideal in certain circumstances, such as processing time-sensitive or highly confidential data, or when faced with network bandwidth constraints, among other examples. Accordingly, in some embodiments, visual fog system **100** may leverage "edge" processing to augment the performance and capabilities of the cloud **130** using edge resources **110**. Edge processing is an approach that involves processing certain data at the network edge (e.g., using edge resources **110**), near where the data is generated, rather than simply funneling large volumes of data to the cloud for processing and storage. Certain data may still be sent to the cloud, as appropriate, such as for deeper analysis and/or long-term storage. Edge processing may be used to complement the shortcomings of cloud-based processing (e.g., when cloud-based processing is inefficient, ineffective, and/or unsecure), and thus improve the handling of the growing volume, variety, and velocity of data generated by IoT devices **114** and/or other edge resources **110**. For example, in some cases, processing data near its source (e.g., in the network edge) rather than in the cloud may improve performance and/or avoid system failures or disasters. Edge processing may also conserve network bandwidth, which may be particularly beneficial when facing bandwidth constraints and/or limited network connectivity.

[0051] In some cases, the collective use of both edge **110** and cloud **130** resources may be referred to as "fog" computing, as functionality of the "cloud" **130** is effectively extended by the edge resources **110**, thus forming a "fog" over the network edge. Moreover, in some embodiments, devices **110** in the "fog" may connect and/or communicate with each other using an interconnection standard or protocol, such as the open interconnect consortium (OIC) standard specification 1.0, released by the Open Connectivity Foundation™ (OCF) on Dec. 23, 2015, which enables devices to discover and connect with each other; Thread, a networking protocol for Internet-of-Things (IoT) devices used in "smart" home automation and similar deployments, developed by an alliance of organizations named the "Thread Group"; the optimized link state routing (OLSR) protocol; and/or the better approach to mobile ad-hoc networking (B.A.T.M.A.N.), among other examples.

[0052] Moreover, in some embodiments, fog computing may be leveraged by visual fog system **100** for large-scale visual computing applications. For example, in some embodiments, the components of visual fog system **100** (e.g., edge resources **110**, cloud resources **130**) may be implemented with some or all aspects of the visual computing functionality described throughout this disclosure in connection with the remaining FIGURES.

[0053] Any, all, or some of the computing devices of system **100** may be adapted to execute any operating system, including Linux or other UNIX-based operating systems, Microsoft Windows, Windows Server, MacOS, Apple iOS, Google Android, or any customized and/or proprietary operating system, along with virtual machines adapted to virtualize execution of a particular operating system.

[0054] While FIG. **1** is described as containing or being associated with a plurality of elements, not all elements illustrated within system **100** of FIG. **1** may be utilized in each alternative implementation of the present disclosure. Additionally, one or more of the elements described in connection with the examples of FIG. **1** may be located external to system **100**, while in other instances, certain elements may be included within or as a portion of one or more of the other described elements, as well as other elements not described in the illustrated implementation. Further, certain elements illustrated in FIG. **1** may be combined with other components, as well as used for alternative or additional purposes in addition to those purposes described herein.

[0055] Additional embodiments associated with the implementation of a visual fog computing system **100** are described further in connection with the remaining FIGURES. Accordingly, it should be appreciated that visual fog system **100** of FIG. **1** may be implemented with any aspects of the embodiments described throughout this disclosure.

[0056] Example Internet-of-Things (IoT) Implementations

[0057] FIGS. **2-5** illustrate examples of Internet-of-Things (IoT) networks and devices that can be used in accordance with embodiments disclosed herein. For example, the operations and functionality described throughout this disclosure may be embodied by an IoT device or machine in the example form of an electronic processing system, within which a set or sequence of instructions may be executed to cause the electronic processing system to perform any one of the methodologies discussed herein, according to an example embodiment. The machine may be an IoT device or an IoT gateway, including a machine embodied by aspects of a personal computer (PC), a tablet PC, a personal digital assistant (PDA), a mobile telephone or smartphone, or any machine capable of executing instructions (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine may be depicted and referenced in the example above, such machine shall also be taken to include any collection of machines that individually or jointly execute a set (or multiple sets) of instructions to perform any one or more of the methodologies discussed herein. Further, these and like examples to a processor-based system shall be taken to include any set of one or more machines that are controlled by or operated by a processor (e.g., a computer) to individually or jointly execute instructions to perform any one or more of the methodologies discussed herein.

[0058] FIG. **2** illustrates an example domain topology for respective internet-of-things (IoT) networks coupled through links to respective gateways. The internet of things (IoT) is a concept in which a large number of computing devices are interconnected to each other and to the Internet to provide functionality and data acquisition at very low levels. Thus, as used herein, an IoT device may include a semiautonomous device performing a function, such as sensing or control, among others, in communication with other IoT devices and a wider network, such as the Internet.

[0059] Often, IoT devices are limited in memory, size, or functionality, allowing larger numbers to be deployed for a similar cost to smaller numbers of larger devices. However, an IoT device may be a smart phone, laptop, tablet, or PC, or other larger device. Further, an IoT device may be a virtual device, such as an application on a smart phone or other computing device. IoT devices may include IoT gateways, used to couple IoT devices to other IoT devices and to cloud applications, for data storage, process control, and the like.

[0060] Networks of IoT devices may include commercial and home automation devices, such as water distribution systems, electric power distribution systems, pipeline control systems, plant control systems, light switches, thermo-

stats, locks, cameras, alarms, motion sensors, and the like. The IoT devices may be accessible through remote computers, servers, and other systems, for example, to control systems or access data.

[0061] The future growth of the Internet and like networks may involve very large numbers of IoT devices. Accordingly, in the context of the techniques discussed herein, a number of innovations for such future networking will address the need for all these layers to grow unhindered, to discover and make accessible connected resources, and to support the ability to hide and compartmentalize connected resources. Any number of network protocols and communications standards may be used, wherein each protocol and standard is designed to address specific objectives. Further, the protocols are part of the fabric supporting human accessible services that operate regardless of location, time or space. The innovations include service delivery and associated infrastructure, such as hardware and software; security enhancements; and the provision of services based on Quality of Service (QoS) terms specified in service level and service delivery agreements. As will be understood, the use of IoT devices and networks, such as those introduced in FIGS. **2-5**, present a number of new challenges in a heterogeneous network of connectivity comprising a combination of wired and wireless technologies.

[0062] FIG. **2** specifically provides a simplified drawing of a domain topology that may be used for a number of internet-of-things (IoT) networks comprising IoT devices **204**, with the IoT networks **256**, **258**, **260**, **262**, coupled through backbone links **202** to respective gateways **254**. For example, a number of IoT devices **204** may communicate with a gateway **254**, and with each other through the gateway **254**. To simplify the drawing, not every IoT device **204**, or communications link (e.g., link **216**, **222**, **228**, or **232**) is labeled. The backbone links **202** may include any number of wired or wireless technologies, including optical networks, and may be part of a local area network (LAN), a wide area network (WAN), or the Internet. Additionally, such communication links facilitate optical signal paths among both IoT devices **204** and gateways **254**, including the use of MUXing/deMUXing components that facilitate interconnection of the various devices.

[0063] The network topology may include any number of types of IoT networks, such as a mesh network provided with the network **256** using Bluetooth low energy (BLE) links **222**. Other types of IoT networks that may be present include a wireless local area network (WLAN) network **258** used to communicate with IoT devices **204** through IEEE 802.11 (Wi-Fi®) links **228**, a cellular network **260** used to communicate with IoT devices **204** through an LTE/LTE-A (4G) or 5G cellular network, and a low-power wide area (LPWA) network **262**, for example, a LPWA network compatible with the LoRaWan specification promulgated by the LoRa alliance, or a IPv6 over Low Power Wide-Area Networks (LPWAN) network compatible with a specification promulgated by the Internet Engineering Task Force (IETF). Further, the respective IoT networks may communicate with an outside network provider (e.g., a tier 2 or tier 3 provider) using any number of communications links, such as an LTE cellular link, an LPWA link, or a link based on the IEEE 802.15.4 standard, such as Zigbee. The respective IoT networks may also operate with use of a variety of network and internet application protocols such as Constrained Application Protocol (CoAP). The respective IoT networks may also be integrated with coordinator devices that provide a chain of links that forms cluster tree of linked devices and networks.

[0064] Each of these IoT networks may provide opportunities for new technical features, such as those as described herein. The improved technologies and networks may enable the exponential growth of devices and networks, including the use of IoT networks into as fog devices or systems. As the use of such improved technologies grows, the IoT networks may be developed for self-management, functional evolution, and collaboration, without needing direct human intervention. The improved technologies may even enable IoT networks to function without centralized controlled systems. Accordingly, the improved technologies described herein may be used to automate and enhance network management and operation functions far beyond current implementations.

[0065] In an example, communications between IoT devices **204**, such as over the backbone links **202**, may be protected by a decentralized system for authentication, authorization, and accounting (AAA). In a decentralized AAA system, distributed payment, credit, audit, authorization, and authentication systems may be implemented across interconnected heterogeneous network infrastructure. This allows systems and networks to move towards autonomous operations. In these types of autonomous operations, machines may even contract for human resources and negotiate partnerships with other machine networks. This may allow the achievement of mutual objectives and balanced service delivery against outlined, planned service level agreements as well as achieve solutions that provide metering, measurements, traceability and trackability. The creation of new supply chain structures and methods may enable a multitude of services to be created, mined for value, and collapsed without any human involvement.

[0066] Such IoT networks may be further enhanced by the integration of sensing technologies, such as sound, light, electronic traffic, facial and pattern recognition, smell, vibration, into the autonomous organizations among the IoT devices. The integration of sensory systems may allow systematic and autonomous communication and coordination of service delivery against contractual service objectives, orchestration and quality of service (QoS) based swarming and fusion of resources. Some of the individual examples of network-based resource processing include the following.

[0067] The mesh network **256**, for instance, may be enhanced by systems that perform inline data-to-information transforms. For example, self-forming chains of processing resources comprising a multi-link network may distribute the transformation of raw data to information in an efficient manner, and the ability to differentiate between assets and resources and the associated management of each. Furthermore, the proper components of infrastructure and resource based trust and service indices may be inserted to improve the data integrity, quality, assurance and deliver a metric of data confidence.

[0068] The WLAN network **258**, for instance, may use systems that perform standards conversion to provide multi-standard connectivity, enabling IoT devices **204** using different protocols to communicate. Further systems may provide seamless interconnectivity across a multi-standard infrastructure comprising visible Internet resources and hidden Internet resources.

[0069] Communications in the cellular network **260**, for instance, may be enhanced by systems that offload data, extend communications to more remote devices, or both. The LPWA network **262** may include systems that perform non-Internet protocol (IP) to IP interconnections, addressing, and routing. Further, each of the IoT devices **204** may include the appropriate transceiver for wide area communications with that device. Further, each IoT device **204** may include other transceivers for communications using additional protocols and frequencies.

[0070] Finally, clusters of IoT devices may be equipped to communicate with other IoT devices as well as with a cloud network. This may allow the IoT devices to form an ad-hoc network between the devices, allowing them to function as a single device, which may be termed a fog device. This configuration is discussed further with respect to FIG. **3** below.

[0071] FIG. **3** illustrates a cloud computing network in communication with a mesh network of IoT devices (devices **302**) operating as a fog device at the edge of the cloud computing network. The mesh network of IoT devices may be termed a fog **320**, operating at the edge of the cloud **300**. To simplify the diagram, not every IoT device **302** is labeled.

[0072] The fog **320** may be considered to be a massively interconnected network wherein a number of IoT devices **302** are in communications with each other, for example, by radio links **322**. As an example, this interconnected network may be facilitated using an interconnect specification released by the Open Connectivity Foundation™ (OCF). This standard allows devices to discover each other and establish communications for interconnects. Other interconnection protocols may also be used, including, for example, the optimized link state routing (OLSR) Protocol, the better approach to mobile ad-hoc networking (B.A.T.M.A.N.) routing protocol, or the OMA Lightweight M2M (LWM2M) protocol, among others.

[0073] Three types of IoT devices **302** are shown in this example, gateways **304**, data aggregators **326**, and sensors **328**, although any combinations of IoT devices **302** and functionality may be used. The gateways **304** may be edge devices that provide communications between the cloud **300** and the fog **320**, and may also provide the backend process function for data obtained from sensors **328**, such as motion data, flow data, temperature data, and the like. The data aggregators **326** may collect data from any number of the sensors **328**, and perform the back-end processing function for the analysis. The results, raw data, or both may be passed along to the cloud **300** through the gateways **304**. The sensors **328** may be full IoT devices **302**, for example, capable of both collecting data and processing the data. In some cases, the sensors **328** may be more limited in functionality, for example, collecting the data and allowing the data aggregators **326** or gateways **304** to process the data.

[0074] Communications from any IoT device **302** may be passed along a convenient path (e.g., a most convenient path) between any of the IoT devices **302** to reach the gateways **304**. In these networks, the number of interconnections provide substantial redundancy, allowing communications to be maintained, even with the loss of a number of IoT devices **302**. Further, the use of a mesh network may allow IoT devices **302** that are very low power or located at a distance from infrastructure to be used, as the range to connect to another IoT device **302** may be much less than the range to connect to the gateways **304**.

[0075] The fog **320** provided from these IoT devices **302** may be presented to devices in the cloud **300**, such as a server **306**, as a single device located at the edge of the cloud **300**, e.g., a fog device. In this example, the alerts coming from the fog device may be sent without being identified as coming from a specific IoT device **302** within the fog **320**. In this fashion, the fog **320** may be considered a distributed platform that provides computing and storage resources to perform processing or data-intensive tasks such as data analytics, data aggregation, and machine-learning, among others.

[0076] In some examples, the IoT devices **302** may be configured using an imperative programming style, e.g., with each IoT device **302** having a specific function and communication partners. However, the IoT devices **302** forming the fog device may be configured in a declarative programming style, allowing the IoT devices **302** to reconfigure their operations and communications, such as to determine needed resources in response to conditions, queries, and device failures. As an example, a query from a user located at a server **306** about the operations of a subset of equipment monitored by the IoT devices **302** may result in the fog **320** device selecting the IoT devices **302**, such as particular sensors **328**, needed to answer the query. The data from these sensors **328** may then be aggregated and analyzed by any combination of the sensors **328**, data aggregators **326**, or gateways **304**, before being sent on by the fog **320** device to the server **306** to answer the query. In this example, IoT devices **302** in the fog **320** may select the sensors **328** used based on the query, such as adding data from flow sensors or temperature sensors. Further, if some of the IoT devices **302** are not operational, other IoT devices **302** in the fog **320** device may provide analogous data, if available.

[0077] FIG. **4** illustrates a drawing of a cloud computing network, or cloud **400**, in communication with a number of Internet of Things (IoT) devices. The cloud **400** may represent the Internet, or may be a local area network (LAN), or a wide area network (WAN), such as a proprietary network for a company. The IoT devices may include any number of different types of devices, grouped in various combinations. For example, a traffic control group **406** may include IoT devices along streets in a city. These IoT devices may include stoplights, traffic flow monitors, cameras, weather sensors, and the like. The traffic control group **406**, or other subgroups, may be in communication with the cloud **400** through wired or wireless links **408**, such as LPWA links, optical links, and the like. Further, a wired or wireless sub-network **412** may allow the IoT devices to communicate with each other, such as through a local area network, a wireless local area network, and the like. The IoT devices may use another device, such as a gateway **510** or **528** to communicate with remote locations such as the cloud **500**; the IoT devices may also use one or more servers **530** to facilitate communication with the cloud **500** or with the gateway **510**. For example, the one or more servers **530** may operate as an intermediate network node to support a local edge cloud or fog implementation among a local area network. Further, the gateway **528** that is depicted may operate in a cloud-to-gateway-to-many edge devices configuration, such as with the various IoT devices **514**, **520**, **524** being constrained or dynamic to an assignment and use of resources in the cloud **500**.

[0078] Other example groups of IoT devices may include remote weather stations **414**, local information terminals **416**, alarm systems **418**, automated teller machines **420**, alarm panels **422**, or moving vehicles, such as emergency vehicles **424** or other vehicles **426**, among many others. Each of these IoT devices may be in communication with other IoT devices, with servers **404**, with another IoT fog device or system (not shown, but depicted in FIG. **3**), or a combination therein. The groups of IoT devices may be deployed in various residential, commercial, and industrial settings (including in both private or public environments).

[0079] As can be seen from FIG. **4**, a large number of IoT devices may be communicating through the cloud **400**. This may allow different IoT devices to request or provide information to other devices autonomously. For example, a group of IoT devices (e.g., the traffic control group **406**) may request a current weather forecast from a group of remote weather stations **414**, which may provide the forecast without human intervention. Further, an emergency vehicle **424** may be alerted by an automated teller machine **420** that a burglary is in progress. As the emergency vehicle **424** proceeds towards the automated teller machine **420**, it may access the traffic control group **406** to request clearance to the location, for example, by lights turning red to block cross traffic at an intersection in sufficient time for the emergency vehicle **424** to have unimpeded access to the intersection.

[0080] Clusters of IoT devices, such as the remote weather stations **414** or the traffic control group **406**, may be equipped to communicate with other IoT devices as well as with the cloud **400**. This may allow the IoT devices to form an ad-hoc network between the devices, allowing them to function as a single device, which may be termed a fog device or system (e.g., as described above with reference to FIG. **3**).

[0081] FIG. **5** is a block diagram of an example of components that may be present in an IoT device **550** for implementing the techniques described herein. The IoT device **550** may include any combinations of the components shown in the example or referenced in the disclosure above. The components may be implemented as ICs, portions thereof, discrete electronic devices, or other modules, logic, hardware, software, firmware, or a combination thereof adapted in the IoT device **550**, or as components otherwise incorporated within a chassis of a larger system. Additionally, the block diagram of FIG. **5** is intended to depict a high-level view of components of the IoT device **550**. However, some of the components shown may be omitted, additional components may be present, and different arrangement of the components shown may occur in other implementations.

[0082] The IoT device **550** may include a processor **552**, which may be a microprocessor, a multi-core processor, a multithreaded processor, an ultra-low voltage processor, an embedded processor, or other known processing element. The processor **552** may be a part of a system on a chip (SoC) in which the processor **552** and other components are formed into a single integrated circuit, or a single package, such as the Edison™ or Galileo™ SoC boards from Intel. As an example, the processor **552** may include an Intel® Architecture Core™ based processor, such as a Quark™, an Atom™, an i3, an i5, an i7, or an MCU-class processor, or another such processor available from Intel® Corporation, Santa Clara, Calif. However, any number other processors may be used, such as available from Advanced Micro Devices, Inc. (AMD) of Sunnyvale, Calif., a MIPS-based design from MIPS Technologies, Inc. of Sunnyvale, Calif., an ARM-based design licensed from ARM Holdings, Ltd. or customer thereof, or their licensees or adopters. The processors may include units such as an A5-A10 processor from Apple® Inc., a Snapdragon™ processor from Qualcomm® Technologies, Inc., or an OMAP™ processor from Texas Instruments, Inc.

[0083] The processor **552** may communicate with a system memory **554** over an interconnect **556** (e.g., a bus). Any number of memory devices may be used to provide for a given amount of system memory. As examples, the memory may be random access memory (RAM) in accordance with a Joint Electron Devices Engineering Council (JEDEC) design such as the DDR or mobile DDR standards (e.g., LPDDR, LPDDR2, LPDDR3, or LPDDR4). In various implementations, the individual memory devices may be of any number of different package types such as single die package (SDP), dual die package (DDP) or quad die package (Q17P). These devices, in some examples, may be directly soldered onto a motherboard to provide a lower profile solution, while in other examples the devices are configured as one or more memory modules that in turn couple to the motherboard by a given connector. Any number of other memory implementations may be used, such as other types of memory modules, e.g., dual inline memory modules (DIMMs) of different varieties including but not limited to microDIMMs or MiniDIMMs.

[0084] To provide for persistent storage of information such as data, applications, operating systems and so forth, a storage **558** may also couple to the processor **552** via the interconnect **556**. In an example, the storage **558** may be implemented via a solid state disk drive (SSDD). Other devices that may be used for the storage **558** include flash memory cards, such as SD cards, microSD cards, xD picture cards, and the like, and USB flash drives. In low power implementations, the storage **558** may be on-die memory or registers associated with the processor **552**. However, in some examples, the storage **558** may be implemented using a micro hard disk drive (HDD). Further, any number of new technologies may be used for the storage **558** in addition to, or instead of, the technologies described, such resistance change memories, phase change memories, holographic memories, or chemical memories, among others.

[0085] The components may communicate over the interconnect **556**. The interconnect **556** may include any number of technologies, including industry standard architecture (ISA), extended ISA (EISA), peripheral component interconnect (PCI), peripheral component interconnect extended (PCIx), PCI express (PCIe), or any number of other technologies. The interconnect **556** may be a proprietary bus, for example, used in a SoC based system. Other bus systems may be included, such as an I2C interface, an SPI interface, point to point interfaces, and a power bus, among others.

[0086] The interconnect **556** may couple the processor **552** to a mesh transceiver **562**, for communications with other mesh devices **564**. The mesh transceiver **562** may use any number of frequencies and protocols, such as 2.4 Gigahertz (GHz) transmissions under the IEEE 802.15.4 standard, using the Bluetooth® low energy (BLE) standard, as defined by the Bluetooth® Special Interest Group, or the ZigBee© standard, among others. Any number of radios, configured for a particular wireless communication protocol, may be used for the connections to the mesh devices **564**. For

example, a WLAN unit may be used to implement Wi-Fi™ communications in accordance with the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard. In addition, wireless wide area communications, e.g., according to a cellular or other wireless wide area protocol, may occur via a WWAN unit.

[0087] The mesh transceiver **562** may communicate using multiple standards or radios for communications at different range. For example, the IoT device **550** may communicate with close devices, e.g., within about 10 meters, using a local transceiver based on BLE, or another low power radio, to save power. More distant mesh devices **564**, e.g., within about 50 meters, may be reached over ZigBee or other intermediate power radios. Both communications techniques may take place over a single radio at different power levels, or may take place over separate transceivers, for example, a local transceiver using BLE and a separate mesh transceiver using ZigBee.

[0088] A wireless network transceiver **566** may be included to communicate with devices or services in the cloud **500** via local or wide area network protocols. The wireless network transceiver **566** may be a LPWA transceiver that follows the IEEE 802.15.4, or IEEE 802.15.4g standards, among others. The IoT device **550** may communicate over a wide area using LoRaWAN™ (Long Range Wide Area Network) developed by Semtech and the LoRa Alliance. The techniques described herein are not limited to these technologies, but may be used with any number of other cloud transceivers that implement long range, low bandwidth communications, such as Sigfox, and other technologies. Further, other communications techniques, such as time-slotted channel hopping, described in the IEEE 802.15.4e specification may be used.

[0089] Any number of other radio communications and protocols may be used in addition to the systems mentioned for the mesh transceiver **562** and wireless network transceiver **566**, as described herein. For example, the radio transceivers **562** and **566** may include an LTE or other cellular transceiver that uses spread spectrum (SPA/SAS) communications for implementing high speed communications. Further, any number of other protocols may be used, such as Wi-Fi® networks for medium speed communications and provision of network communications.

[0090] The radio transceivers **562** and **566** may include radios that are compatible with any number of 3GPP (Third Generation Partnership Project) specifications, notably Long Term Evolution (LTE), Long Term Evolution-Advanced (LTE-A), and Long Term Evolution-Advanced Pro (LTE-A Pro). It can be noted that radios compatible with any number of other fixed, mobile, or satellite communication technologies and standards may be selected. These may include, for example, any Cellular Wide Area radio communication technology, which may include e.g. a 5th Generation (5G) communication systems, a Global System for Mobile Communications (GSM) radio communication technology, a General Packet Radio Service (GPRS) radio communication technology, or an Enhanced Data Rates for GSM Evolution (EDGE) radio communication technology, a UMTS (Universal Mobile Telecommunications System) communication technology, In addition to the standards listed above, any number of satellite uplink technologies may be used for the wireless network transceiver **566**, including, for example, radios compliant with standards issued by the ITU (International Telecommunication Union), or the ETSI (European Telecommunications Standards Institute), among others. The examples provided herein are thus understood as being applicable to various other communication technologies, both existing and not yet formulated.

[0091] A network interface controller (NIC) **568** may be included to provide a wired communication to the cloud **500** or to other devices, such as the mesh devices **564**. The wired communication may provide an Ethernet connection, or may be based on other types of networks, such as Controller Area Network (CAN), Local Interconnect Network (LIN), DeviceNet, ControlNet, Data Highway+, PROFIBUS, or PROFINET, among many others. An additional NIC **568** may be included to allow connect to a second network, for example, a NIC **568** providing communications to the cloud over Ethernet, and a second NIC **568** providing communications to other devices over another type of network.

[0092] The interconnect **556** may couple the processor **552** to an external interface **570** that is used to connect external devices or subsystems. The external devices may include sensors **572**, such as accelerometers, level sensors, flow sensors, optical light sensors, camera sensors, temperature sensors, a global positioning system (GPS) sensors, pressure sensors, barometric pressure sensors, and the like. The external interface **570** further may be used to connect the IoT device **550** to actuators **574**, such as power switches, valve actuators, an audible sound generator, a visual warning device, and the like.

[0093] In some optional examples, various input/output (I/O) devices may be present within, or connected to, the IoT device **550**. For example, a display or other output device **584** may be included to show information, such as sensor readings or actuator position. An input device **586**, such as a touch screen or keypad may be included to accept input. An output device **584** may include any number of forms of audio or visual display, including simple visual outputs such as binary status indicators (e.g., LEDs) and multi-character visual outputs, or more complex outputs such as display screens (e.g., LCD screens), with the output of characters, graphics, multimedia objects, and the like being generated or produced from the operation of the IoT device **550**.

[0094] A battery **576** may power the IoT device **550**, although in examples in which the IoT device **550** is mounted in a fixed location, it may have a power supply coupled to an electrical grid. The battery **576** may be a lithium ion battery, or a metal-air battery, such as a zinc-air battery, an aluminum-air battery, a lithium-air battery, and the like.

[0095] A battery monitor/charger **578** may be included in the IoT device **550** to track the state of charge (SoCh) of the battery **576**. The battery monitor/charger **578** may be used to monitor other parameters of the battery **576** to provide failure predictions, such as the state of health (SoH) and the state of function (SoF) of the battery **576**. The battery monitor/charger **578** may include a battery monitoring integrated circuit, such as an LTC4020 or an LTC2990 from Linear Technologies, an ADT7488A from ON Semiconductor of Phoenix Ariz., or an IC from the UCD90xxx family from Texas Instruments of Dallas, Tex. The battery monitor/charger **578** may communicate the information on the battery **576** to the processor **552** over the interconnect **556**. The battery monitor/charger **578** may also include an analog-to-digital (ADC) convertor that allows the processor **552** to directly monitor the voltage of the battery **576** or the current flow from the battery **576**. The battery parameters may be

used to determine actions that the IoT device **550** may perform, such as transmission frequency, mesh network operation, sensing frequency, and the like.

[0096] A power block **580**, or other power supply coupled to a grid, may be coupled with the battery monitor/charger **578** to charge the battery **576**. In some examples, the power block **580** may be replaced with a wireless power receiver to obtain the power wirelessly, for example, through a loop antenna in the IoT device **550**. A wireless battery charging circuit, such as an LTC4020 chip from Linear Technologies of Milpitas, Calif., among others, may be included in the battery monitor/charger **578**. The specific charging circuits chosen depend on the size of the battery **576**, and thus, the current required. The charging may be performed using the Airfuel standard promulgated by the Airfuel Alliance, the Qi wireless charging standard promulgated by the Wireless Power Consortium, or the Rezence charging standard, promulgated by the Alliance for Wireless Power, among others.

[0097] The storage **558** may include instructions **582** in the form of software, firmware, or hardware commands to implement the techniques described herein. Although such instructions **582** are shown as code blocks included in the memory **554** and the storage **558**, it may be understood that any of the code blocks may be replaced with hardwired circuits, for example, built into an application specific integrated circuit (ASIC).

[0098] In an example, the instructions **582** provided via the memory **554**, the storage **558**, or the processor **552** may be embodied as a non-transitory, machine readable medium **560** including code to direct the processor **552** to perform electronic operations in the IoT device **550**. The processor **552** may access the non-transitory, machine readable medium **560** over the interconnect **556**. For instance, the non-transitory, machine readable medium **560** may include storage units such as optical disks, flash drives, or any number of other hardware devices. The non-transitory, machine readable medium **560** may include instructions to direct the processor **552** to perform a specific sequence or flow of actions, for example, as described with respect to the flowchart(s) and diagram(s) of operations and functionality described throughout this disclosure.

[0099] Example Computing Architectures

[0100] FIGS. **6** and **7** illustrate example computer processor architectures that can be used in accordance with embodiments disclosed herein. For example, in various embodiments, the computer architectures of FIGS. **6** and **7** may be used to implement the visual fog functionality described throughout this disclosure. Other embodiments may use other processor and system designs and configurations known in the art, for example, for laptops, desktops, handheld PCs, personal digital assistants, engineering workstations, servers, network devices, network hubs, switches, embedded processors, digital signal processors (DSPs), graphics devices, video game devices, set-top boxes, micro controllers, cell phones, portable media players, hand held devices, and various other electronic devices, are also suitable. In general, a huge variety of systems or electronic devices capable of incorporating a processor and/or other execution logic as disclosed herein are generally suitable.

[0101] FIG. **6** illustrates a block diagram for an example embodiment of a processor **600**. Processor **600** is an example of a type of hardware device that can be used in connection with the embodiments described throughout this disclosure. Processor **600** may be any type of processor, such as a microprocessor, an embedded processor, a digital signal processor (DSP), a network processor, a multi-core processor, a single core processor, or other device to execute code. Although only one processor **600** is illustrated in FIG. **6**, a processing element may alternatively include more than one of processor **600** illustrated in FIG. **6**. Processor **600** may be a single-threaded core or, for at least one embodiment, the processor **600** may be multithreaded in that it may include more than one hardware thread context (or "logical processor") per core.

[0102] FIG. **6** also illustrates a memory **602** coupled to processor **600** in accordance with an embodiment. Memory **602** may be any of a wide variety of memories (including various layers of memory hierarchy) as are known or otherwise available to those of skill in the art. Such memory elements can include, but are not limited to, random access memory (RAM), read only memory (ROM), logic blocks of a field programmable gate array (FPGA), erasable programmable read only memory (EPROM), and electrically erasable programmable ROM (EEPROM).

[0103] Processor **600** can execute any type of instructions associated with algorithms, processes, or operations detailed herein. Generally, processor **600** can transform an element or an article (e.g., data) from one state or thing to another state or thing.

[0104] Code **604**, which may be one or more instructions to be executed by processor **600**, may be stored in memory **602**, or may be stored in software, hardware, firmware, or any suitable combination thereof, or in any other internal or external component, device, element, or object where appropriate and based on particular needs. In one example, processor **600** can follow a program sequence of instructions indicated by code **604**. Each instruction enters a front-end logic **606** and is processed by one or more decoders **608**. The decoder may generate, as its output, a micro operation such as a fixed width micro operation in a predefined format, or may generate other instructions, microinstructions, or control signals that reflect the original code instruction. Front-end logic **606** may also include register renaming logic and scheduling logic, which generally allocate resources and queue the operation corresponding to the instruction for execution.

[0105] Processor **600** can also include execution logic **614** having a set of execution units **616***a*, **616***b*, **616***n*, etc. Some embodiments may include a number of execution units dedicated to specific functions or sets of functions. Other embodiments may include only one execution unit or one execution unit that can perform a particular function. Execution logic **614** performs the operations specified by code instructions.

[0106] After completion of execution of the operations specified by the code instructions, back-end logic **618** can retire the instructions of code **604**. In one embodiment, processor **600** allows out of order execution but requires in order retirement of instructions. Retirement logic **620** may take a variety of known forms (e.g., re-order buffers or the like). In this manner, processor **600** is transformed during execution of code **604**, at least in terms of the output generated by the decoder, hardware registers and tables utilized by register renaming logic **610**, and any registers (not shown) modified by execution logic **614**.

[0107] Although not shown in FIG. **6**, a processing element may include other elements on a chip with processor **600**. For example, a processing element may include

memory control logic along with processor **600**. The processing element may include I/O control logic and/or may include I/O control logic integrated with memory control logic. The processing element may also include one or more caches. In some embodiments, non-volatile memory (such as flash memory or fuses) may also be included on the chip with processor **600**.

[0108] FIG. **7** illustrates a block diagram for an example embodiment of a multiprocessor **700**. As shown in FIG. **7**, multiprocessor system **700** is a point-to-point interconnect system, and includes a first processor **770** and a second processor **780** coupled via a point-to-point interconnect **750**. In some embodiments, each of processors **770** and **780** may be some version of processor **600** of FIG. **6**.

[0109] Processors **770** and **780** are shown including integrated memory controller (IMC) units **772** and **782**, respectively. Processor **770** also includes as part of its bus controller units point-to-point (P-P) interfaces **776** and **778**; similarly, second processor **780** includes P-P interfaces **786** and **788**. Processors **770**, **780** may exchange information via a point-to-point (P-P) interface **750** using P-P interface circuits **778**, **788**. As shown in FIG. **7**, IMCs **772** and **782** couple the processors to respective memories, namely a memory **732** and a memory **734**, which may be portions of main memory locally attached to the respective processors.

[0110] Processors **770**, **780** may each exchange information with a chipset **790** via individual P-P interfaces **752**, **754** using point to point interface circuits **776**, **794**, **786**, **798**. Chipset **790** may optionally exchange information with the coprocessor **738** via a high-performance interface **739**. In one embodiment, the coprocessor **738** is a special-purpose processor, such as, for example, a high-throughput MIC processor, a network or communication processor, compression engine, graphics processor, GPGPU, embedded processor, matrix processor, or the like.

[0111] A shared cache (not shown) may be included in either processor or outside of both processors, yet connected with the processors via P-P interconnect, such that either or both processors' local cache information may be stored in the shared cache if a processor is placed into a low power mode.

[0112] Chipset **790** may be coupled to a first bus **716** via an interface **796**. In one embodiment, first bus **716** may be a Peripheral Component Interconnect (PCI) bus, or a bus such as a PCI Express bus or another third generation I/O interconnect bus, although the scope of this disclosure is not so limited.

[0113] As shown in FIG. **7**, various I/O devices **714** may be coupled to first bus **716**, along with a bus bridge **718** which couples first bus **716** to a second bus **720**. In one embodiment, one or more additional processor(s) **715**, such as coprocessors, high-throughput MIC processors, GPGPU's, accelerators (such as, e.g., graphics accelerators or digital signal processing (DSP) units), matrix processors, field programmable gate arrays, or any other processor, are coupled to first bus **716**. In one embodiment, second bus **720** may be a low pin count (LPC) bus. Various devices may be coupled to a second bus **720** including, for example, a keyboard and/or mouse **722**, communication devices **727** and a storage unit **728** such as a disk drive or other mass storage device which may include instructions/code and data **730**, in one embodiment. Further, an audio I/O **724** may be coupled to the second bus **720**. Note that other architectures are possible. For example, instead of the point-to-point architecture of FIG. **7**, a system may implement a multi-drop bus or other such architecture.

[0114] All or part of any component of FIG. **7** may be implemented as a separate or stand-alone component or chip, or may be integrated with other components or chips, such as a system-on-a-chip (SoC) that integrates various computer components into a single chip.

[0115] Embodiments of the mechanisms disclosed herein may be implemented in hardware, software, firmware, or a combination of such implementation approaches. Certain embodiments may be implemented as computer programs or program code executing on programmable systems comprising at least one processor, a storage system (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device.

[0116] Program code, such as code **730** illustrated in FIG. **7**, may be applied to input instructions to perform the functions described herein and generate output information. The output information may be applied to one or more output devices, in known fashion. For purposes of this application, a processing system includes any system that has a processor, such as, for example; a digital signal processor (DSP), a microcontroller, an application specific integrated circuit (ASIC), or a microprocessor.

[0117] The program code may be implemented in a high level procedural or object oriented programming language to communicate with a processing system. The program code may also be implemented in assembly or machine language, if desired. In fact, the mechanisms described herein are not limited in scope to any particular programming language. In any case, the language may be a compiled or interpreted language.

[0118] One or more aspects of at least one embodiment may be implemented by representative instructions stored on a machine-readable medium which represents various logic within the processor, which when read by a machine causes the machine to fabricate logic to perform the techniques described herein. Such representations, known as "IP cores" may be stored on a tangible, machine readable medium and supplied to various customers or manufacturing facilities to load into the fabrication machines that actually make the logic or processor.

[0119] Such machine-readable storage media may include, without limitation, non-transitory, tangible arrangements of articles manufactured or formed by a machine or device, including storage media such as hard disks, any other type of disk including floppy disks, optical disks, compact disk read-only memories (CD-ROMs), compact disk rewritable's (CD-RWs), and magneto-optical disks, semiconductor devices such as read-only memories (ROMs), random access memories (RAMs) such as dynamic random access memories (DRAMs), static random access memories (SRAMs), erasable programmable read-only memories (EPROMs), flash memories, electrically erasable programmable read-only memories (EEPROMs), phase change memory (PCM), magnetic or optical cards, or any other type of media suitable for storing electronic instructions.

[0120] Accordingly, embodiments of this disclosure also include non-transitory, tangible machine-readable media containing instructions or containing design data, such as Hardware Description Language (HDL), which defines structures, circuits, apparatuses, processors and/or system features described herein. Such embodiments may also be referred to as program products.

[0121] Visual Fog Architecture

[0122] FIG. **8** illustrates an example embodiment of an architecture **800** for visual fog nodes. In some embodiments, for example, fog node architecture **800** may be used to implement the functionality of fog nodes **810** in a visual fog network or system (e.g., visual fog system **100** of FIG. **1**). A fog node **810**, for example, can include any node or component that ranges from the edge of a network to the cloud, inclusively.

[0123] In the illustrated embodiment, fog node **810** includes various application programming interfaces (APIs) that provide fundamental capabilities for fog node **810**, such as auxiliary API **820**, primitive vision API **830**, and storage API **840**. In some embodiments, for example, these APIs may be used or implemented by lower-level algorithm developers.

[0124] Auxiliary API **820** provides various fundamental functionality for fog node **810**, such as security **822***a*, communication **822***b*, compression **822***c* (e.g., codecs), and so forth.

[0125] Primitive vision API **830** provides fundamental vision processing capabilities for fog node **810**. For example, primitive vision API **830** provides access to a plurality of vision kernels **832** that can be used to perform primitive vision operations (e.g., person or object detection, facial recognition). Primitive vision API **830** may also provide access to various machine learning and/or neural network frameworks (e.g., Caffe, OpenCV, TensorFlow).

[0126] Storage API **840** provides storage capabilities for fog node **810**. In some embodiments, for example, storage API **840** may include a variety of databases **842** for storing different types of visual data, such as graph databases, relational databases, array-based databases (e.g., TileDB), and so forth. In some embodiments, for example, the particular database used to store certain visual data may depend on the type of data, such as raw visual data or pixels, compressed visual data, visual metadata, and so forth.

[0127] Moreover, fog node **810** further includes a vision application API **850** that provides higher-level vision functionality, which may be used or implemented by developers of vision applications. For example, vision application API **850** may include a privacy policy **852** that defines the requisite privacy treatment for all data and devices associated with a visual fog network. Vision application API **850** may also include a vision kernel management service **854** that provides access to a variety of primitive vision operations or vision kernels. In some embodiments, for example, vision kernel management service **854** may retrieve vision kernels from a vision kernel repository. For example, if a particular vision application employs person detection functionality, vision kernel management service **854** may retrieve the appropriate vision kernel for performing person detection using the available hardware of the particular fog node **810**.

[0128] Fog node **810** further includes a vision analytics API **860** and query API **870**, which may be used by end-users or operators to perform visual analytics and visual queries. For example, vision analytics API **860** may perform inline (e.g. real-time) and/or offline processing of visual data, application launching, scheduling, resource monitoring, and so forth. Vision analytics API **860** may also include a vision application management service **862** that provides access to a variety of vision applications (e.g., people searching/tracking, object detection/tracking, and so forth). In some embodiments, for example, vision application management service **862** may retrieve vision applications from a vision application repository. In this manner, if an end-user wants to perform a people search, vision application management service **862** may retrieve an appropriate vision application for people searching. In some embodiments, for example, a people search vision application may use vision kernels that perform person detection followed by facial recognition. The end-user, however, can utilize the people search vision application without any knowledge of the underlying vision kernels or vision operations used to implement the application.

[0129] Moreover, query API **870** provides an interface that enables end-users to submit visual search requests or queries. In some embodiments, for example, query API **870** may support flexible visual queries in a variety of syntaxes, such as natural language, functional syntax (e.g., using logical operators), relational syntax, and so forth. In some embodiments, query API **870** may further include a query primitive repository **874** that contains the primitive operations that are supported for visual queries. Moreover, query API **870** may include a query compiler **872** for compiling the visual queries into visual processing dataflows that can be executed by visual fog nodes.

[0130] FIG. **9-12** illustrate example embodiments of visual fog architectures.

[0131] For example, FIG. **9** illustrates an example visual fog architecture **900** that includes cameras **902**, sensors **904**, local analytics framework **906**, inline analytics framework **908**, offline analytics framework **910**, storage **912**, and presentation/interpretation framework **914**. In the illustrated embodiment, for example, cameras **902** and/or sensors **904** may generate visual data, such as images and/or video. The visual data may then be provided to local analytics framework **906**, which may be used to perform preliminary processing and analytics at the network edge (e.g., near the cameras **902** or sensors **904** that captured the visual data). The partially processed visual data may then be provided to inline analytics framework **908** for further processing in real-time. In various embodiments, for example, inline analytics may be performed by and/or distributed across any combination of fog devices or resources (e.g., mobile devices, IoT devices, gateways, and/or the cloud). The resulting visual data and/or metadata from inline analytics framework **908** may then be stored in data storage **912**. Moreover, a visual search query may be subsequently received by presentation/interpretation framework **914** (e.g., from an end-user). Accordingly, presentation/interpretation framework **914** may interact with data storage **912** and/or inline analytics framework **908** to determine whether a response to the query can be formulated based on the visual data and/or metadata that has already been processed or generated. If further processing needs to be performed to respond to the query, however, presentation/interpretation framework **914** may interact with offline analytics framework **910** to perform further offline processing of the visual data. In various embodiments, for example, offline analytics may be performed by and/or distributed across any combination of fog devices or resources (e.g., mobile devices, IoT devices, gateways, and/or the cloud). Accordingly, based on the information obtained either from data storage **912**, inline analytics framework **908**, and/or offline analytics framework **910**, presentation/interpretation framework **914** may then respond to the visual query.

[0132] FIG. **10** illustrates an example visual processing pipeline **1000** associated with a visual fog architecture. In the illustrated example, visual data **1002** may first be captured by cameras and/or visual sensors, and the visual data **1002** may then be processed to perform certain visual functions **1004** (e.g., face detection) and/or other analytics, resulting in a set of visual metadata **1012** that may be stored in data storage **1010**. Moreover, an end-user may subsequently submit an ad hoc search query **1006** associated with the visual data **1002**, and a query compiler/interpreter **1008** may then compile the query into a visual processing dataflow that can be executed (e.g., using available fog nodes or resources) in order to respond to the query. In some cases, for example, it may be possible to formulate a query result **1016** based on the processing that has already been completed. For example, in some cases, the query result **1016** may be formulated by applying appropriate logic operations **1014** on the existing visual metadata **1012** that has already been generated. In other cases, however, further visual processing and/or functions **1004** may need to be performed on the visual data **1002** in order to formulate the query result **1016**. In either case, the compiler/interpreter **1008** may generate a requisite vision processing dataflow for responding to the query, and the resulting vision processing dataflow may then be executed in order to formulate the query result **1016**.

[0133] FIG. **11** illustrates another example visual fog architecture **1100**. In the illustrated embodiment, visual data captured by cameras **1140***b* is provided to a distributed runtime environment **1120**, which performs initial pre-processing on the visual data in real-time (e.g., when the visual data is first captured rather than in response to a query). The resulting visual data or metadata generated by the distributed runtime environment **1120** is then stored in data storage **1130**.

[0134] Separately, visual search queries containing user-defined vision functions (UVFs) **1104***a-c* are received from end-users **1102** of visual fog **1100**. A UVF **1104** received from an end-user **1102** is first processed by a compiler **1110** in order to generate a vision dataflow graph for executing the UVF. Accordingly, the vision dataflow graph is then executed by the distributed runtime environment **1120** in order to generate a result for the UVF **1104**. In some embodiments, for example, the distributed runtime environment **1120** may determine the result using existing visual metadata that has already been generated (e.g., from the initial or real-time processing of the original visual data), and/or by performing further analysis on the visual data (e.g., by executing a particular vision application **1150**). The result obtained from execution of the UVF **1104** may then be provided back to the requesting end-user **1102**.

[0135] Further, in various embodiments, the distributed runtime environment **1120** may perform the described visual data processing (e.g., initial pre-processing and/or UVF processing) by scheduling or distributing vision workloads across the available fog devices or resources **1140** (e.g., cloud servers **1140***a*, cameras **1140***b*, mobile devices, IoT devices, gateways, and/or other fog/edge devices).

[0136] FIGS. **12**A-B illustrate another example visual fog architecture **1200**. In the illustrated embodiment, visual fog architecture **1200** includes a network of fog devices **1216**, including cameras or visual sensors **1216***a*, gateways **1216***b*, and cloud servers **1216***c*. The cameras or visual sensors **1216***a*, for example, are used to capture visual data **1217**. Moreover, a computer vision expert **1202** can develop an imperative vision program **1203** that leverages the captured visual data **1217**. The vision program **1203**, for example, may be implemented using programming and composability frameworks **1208** and **1210** to define vision processing dataflows **1209** and generate vision processing workloads **1211**.

[0137] In the illustrated embodiment, for example, the vision program **1203** leverages a distributed runtime environment **1214** to process visual data **1217** captured in visual fog **1200**. The distributed runtime environment **1214**, for example, can perform visual data processing using the collection of available fog devices **1216** in visual fog **1200**.

[0138] In some embodiments, for example, the distributed runtime environment **1214** may be used to perform initial pre-processing on captured visual data **1217** in real-time (e.g., when the visual data is first captured rather than in response to a query). The resulting visual data or metadata **1217** generated by the distributed runtime environment **1214** may then be stored in a database or data storage **1218**.

[0139] Moreover, a layperson or end-user **1204** may subsequently submit a declarative query **1205** associated with visual data **1217** captured by visual fog **1200**. The declarative query **1205** is processed by a visual question answering (VQA) system **1206**, which uses a compiler or interpreter to generate a dataflow **1209** for responding to the query. In some cases, for example, it may be possible to respond to query **1205** using existing visual metadata **1217** that has already been generated (e.g., during the initial or real-time processing of the original visual data **1217** and/or during the processing associated with prior queries **1205**). In other cases, however, further processing may need to be performed on the visual data **1217** in order to respond to the query **1205**. In either case, an appropriate dataflow **1209** for responding to the query **1205** may be generated, and the resulting dataflow **1209** may be further partitioned into one or more underlying vision processing workloads **1211**. Moreover, based on the resource availability **1215** of fog devices **1216** in the distributed runtime environment **1214**, a schedule **1213** for distributing the workloads **1211** across the available fog devices **1216** may be generated. Accordingly, the respective workloads **1211** may then be distributed across the fog devices **1216** based on the generated schedule **1213**, and each fog device **1216** may execute its respective workload(s) **1211**. In this manner, the dataflow **1209** for responding to the query **1205** is executed by the various fog devices **1216** using a distributed approach. A response to the query **1205** may then be provided to the end-user **1204**, and the resulting visual metadata **1217** may be stored in database **1218** for responding to subsequent queries.

[0140] Visual Question Answering (VQA)

[0141] FIG. **13-14** illustrate example embodiments associated with a visual question answering (VQA) framework. In some embodiments, for example, a visual fog architecture may implement a VQA framework to provide a flexible and efficient interface for end-users to submit ad hoc visual search queries. In visual processing systems, for example, the ability to submit a query to search large data sets in an efficient manner (e.g., millions of images) and identify a subset of relevant images or related information is important. Existing visual processing solutions are implemented using rigid or inflexible approaches, however, and are unable to search visual data efficiently. Accordingly, the visual

question answering (VQA) framework of FIGS. **13** and **14** can be used to alleviate the deficiencies of existing solutions.

[0142] In some embodiments, for example, a VQA framework may support flexible or ad hoc visual search queries using a variety of syntaxes, such as natural language, functional syntax (e.g., using logical operators), relational syntax, and so forth. Accordingly, when a visual search query is received from a user, the query may be compiled into a visual processing dataflow that can be distributed across and executed by the various fog nodes in a visual fog architecture. In this manner, end-users can perform complex searches on large sets of visual data without any knowledge of the underlying architecture or processing required to execute the searches.

[0143] Moreover, in some embodiments, users or developers may be capable of defining custom vision functions that can be used in visual search queries, referred to as user-defined vision functions (UVFs). As an example, a UVF could be defined for visually equivalency, or performing "equal" operations on visual data. Many ad hoc visual queries, for example, require information related to the same object or person to be identified or grouped together. Identifying the same object or person across different images or video streams, however, can be challenging. In some embodiments, for example, this task may require feature extraction to be performed across multiple cameras. The respective features extracted from each camera often differ, however, and not all cameras have the same field of view, and thus certain features may be successfully extracted from some cameras but not others. Accordingly, in some embodiments, a user may implement a UVF to define how visual equivalency or "equal" operations are to be performed on visual data. In some embodiments, for example, a UVF for visual equivalency may define objects as "equal" if their feature vectors are "close enough" to each other, meaning the feature vectors must be sufficiently similar but do not have to be an exact match. Further, if feature vectors from different cameras are missing certain features, only the partial features will be compared and the "close enough" definition will be scaled accordingly.

[0144] FIG. **13** illustrates an example embodiment of a visual question answering (VQA) pipeline **1300**. In the illustrated example, a visual query **1302** is first received from an end-user, and a dataflow compiler **1304** is then used to compile the visual query **1302** into a visual processing pipeline or dataflow **1308**. In some embodiments, for example, the dataflow compiler **1304** may use a library of vision kernel modules **1306** (e.g., face recognition, pose recognition, object recognition, and so forth) to generate the resulting visual processing dataflow **1308**.

[0145] In some cases, for example, the visual processing dataflow **1308** may leverage existing visual metadata that has already been generated and stored on data storage **1314**. For example, an inline analytics framework **1310** may be used to perform initial visual data processing in real-time (e.g., when visual data is first captured rather than in response to a query), and an offline analytics framework **1312** may be used to perform further visual data processing required for responding to search queries. Moreover, both the inline and offline analytics frameworks **1310**, **1312** may store their resulting visual metadata on data storage **1314** for use in responding to subsequent visual search queries. Accordingly, in some cases, the visual processing dataflow **1308** for a particular query **1302** may leverage existing visual metadata that has already been generated and stored on data storage **1314**. In other cases, however, further processing may be required to respond to the query **1302**, and thus the visual processing dataflow **1308** may leverage the offline analytics framework **1312** to perform additional processing. In either case, the visual processing pipeline or dataflow **1308** generated by compiler **1304** is executed by the runtime environment in order to generate a response to the visual query **1302**.

[0146] FIG. **14** illustrates an example embodiment of a visual question answering (VQA) compiler **1400**. In some embodiments, for example, compiler **1400** may be used to compile VQA queries and/or user-defined vision functions (UVFs) **1402** into visual dataflow graphs **1417** that can be distributed across and executed by the various fog nodes in a visual fog architecture.

[0147] In the illustrated embodiment, for example, UVFs **1402** are provided to the compiler **1400** via a declarative API **1412**. The compiler **1400** may then generate a graph of high-level vision operations **1415** that are required to execute the UVFs **1402**, which may in turn be used to generate a vision dataflow graph **1417**. In some embodiments, for example, the vision dataflow graph **1417** may be a directed acyclic graph (DAG) that represents the visual processing pipeline required to execute the particular UVFs **1402**. Moreover, the compiler **1400** may use dataflow deduplication to optimize the vision dataflow graph **1417**, for example, by merging redundant portions of the dataflows of multiple UVFs **1402** to eliminate the redundancies.

[0148] In some embodiments, for example, compiler **1400** may generate the vision dataflow graph **1417** using information from the underlying vision modules **1418** (e.g., hardware-specific information required for scheduling workloads on heterogeneous hardware). The compiler **1400** may also generate a number of database API calls to obtain visual data and/or metadata required to execute the UVFs **1402**. In various embodiments, these database API calls may either be part of, or separate from, the vision dataflow graph **1417**. Moreover, in some embodiments, the compiler **1400** may generate different results depending on the available visual metadata.

[0149] In this manner, the resulting vision dataflow graph **1417** generated by compiler **1400** can subsequently be executed by the runtime environment in order to generate the results for responding to UVFs **1402**.

[0150] Runtime

[0151] The visual fog paradigm envisions tens of thousands (or more) heterogeneous, camera-enabled edge devices distributed across the Internet and/or other large-scale networks, providing live sensing for a myriad of different visual processing applications, given task parallelism and data parallelism. The scale, computational demands, and bandwidth needed for visual computing pipelines necessitates intelligent offloading to distributed computing infrastructure, including the cloud, Internet gateway devices, and the edge devices themselves.

[0152] In some embodiments, for example, visual processing may be scheduled or distributed across available fog devices based on various criteria, including device connectivity, device resource capabilities, device resource availability, workload type, privacy constraints, and so forth. Further, machine learning can be leveraged to optimize scheduling decisions.

[0153] Workload deployment and/or migration can be implemented using a hot-pluggable runtime environment with universal plugin APIs. For example, conventional workload deployment/migration can be expensive, as it may require sending the runtime environment and toolchains to the assigned nodes. With hot-pluggable runtimes, however, workloads are hot-swappable (e.g., stop runtime, replace plugin, start runtime).

[0154] Moreover, a plugin or vision kernel repository can be used to facilitate workload deployment. For example, a cloud-based or distributed repository may be used to manage a collection of device and implementation abstractions for each supported vision capability. In this manner, the repository can distribute the appropriate plugins or vision kernels to fog nodes based on their respective workload assignments.

[0155] Incremental processing may be leveraged by a visual fog runtime to maintain the state of any prior processing that has already been performed on visual data, enabling the results of the prior processing to be leveraged for subsequent visual processing and queries. For example, the results of any processing performed on visual data may be represented as visual metadata, which may be stored for later use to avoid performing duplicative processing for subsequent visual queries. In this manner, when a visual query or UVF is received, the dataflow generated by a compiler may vary depending on the available metadata that has already been generated and can be reused.

[0156] Metadata pre-provisioning can be used to reduce vision query latency by pre-processing visual data to complete common or frequent types of processing in advance. In some embodiments, for example, a machine learning model may be used to optimize the types of pre-processing that is performed. For example, based on patterns of queries of the same type or that involve similar types of processing, machine learning may be used to model the relationships of diverse queries, while also taking other modalities into account (e.g., weather, traffic). For example, metadata can be pre-provisioned by pre-scheduling certain types of processing in advance based on the recent history of vision queries and UVFs. In this manner, patterns of common or similar vision workloads can trigger pre-processing on newly captured visual data for those types of workloads to reduce query latency.

[0157] Similarly, stream prioritization or prefetching can be used to perform low-latency visual data loading or fetching based on historical trends and/or workflows. For example, the vision processing history can be used to prioritize certain data streams and/or pre-fetch data from memory for a particular application to improve query latency. Compared to metadata pre-provisioning, which involves expedited processing that is performed in advance, stream prioritization involves obtaining or moving visual data to a location where it will likely be needed (e.g., from a camera to certain processing nodes).

[0158] Cached visual analytics can be used to optimize visual processing using cached workflows, similar to incremental processing. For example, based on cached information regarding particular visual streams that have already been obtained and processed, along with the type of processing or workloads performed on those streams, subsequent vision processing dataflows may omit certain processing steps that have previously been performed and whose results have been cached. For example, a visual analytics application involves a number of primitive vision operations. The volume of computation can be reduced, however, by caching visual analytics results and reusing them for subsequent operations when possible. For example, when executing a visual analytics application, cached visual metadata resulting from prior processing can be searched to avoid duplicative computation. In some embodiments, for example, cached visual analytics may be implemented as follows:

[0159] 1. Each primitive vision operation is tagged or labeled using a cache tag;

[0160] 2. For each instance or stream of visual data (e.g., each stored video), any corresponding visual metadata that has already been generated is stored in a metadata database or cache;

[0161] 3. If there is a cache tag hit for a particular primitive vision operation with respect to a particular instance or stream of visual data, then the particular primitive vision operation can be omitted and instead the existing visual metadata can be used; and

[0162] 4. If there is a cache tag miss, however, the particular primitive vision operation is executed and the resulting metadata is cached in the metadata database for subsequent use.

[0163] Tensor factorization can also be used for distributed neural network inferencing in order to address the overfitting problem. For example, representative weights of consecutive neural network layers can utilize tensor factorization to "smooth out" the model.

[0164] FIGS. **15** and **16** illustrate example embodiments of device-centric scheduling for visual fog computing. In some embodiments, for example, visual fog scheduling may depend on (1) device resource capacities, and (2) workload resource requirements. While the former remains constant and consistent, the latter can vary depending on a device's hardware specifications and software toolchains. For example, in some embodiments, there may be multiple implementations of a facial recognition capability that are respectively optimized for different types of hardware, such as CPUs, GPUs, FPGAs, ASICs, and so forth. In this manner, multiple implementations of a single vision capability can be leveraged to create an opportunity for further optimization in visual fog computing.

[0165] Accordingly, in order to address the heterogeneity of devices with different types of hardware and/or software, the illustrated embodiments implement device-centric scheduling using a vision capabilities repository. In some embodiments, for example, the vision capabilities repository may include multiple implementations of a particular vision capability that are optimized for different hardware and/or software environments. In this manner, vision workloads can be scheduled or distributed across fog devices based on their respective types of resources and capabilities, along with per-resource telemetry information that identifies resource availability.

[0166] The basic principle is to abstract capabilities (e.g., face detection, gesture recognition) from their underlying kernels/implementations (e.g., SIFT-based implementations, deep neural network implementations). This type of abstraction provides the flexibility to deploy an arbitrary vision capability on a per-device basis. For example, using resource-based scheduling, heterogeneous resource types of different fog devices can be considered as a whole in order to determine the optimal task-to-device mapping across the

various fog devices, and also identify the corresponding vision capability implementations that each device should use for its assigned tasks. Moreover, resource telemetry can be used to monitor resource availability of fog devices on a per-resource basis (e.g., CPU, GPU, FPGA, ASIC, and so forth) to further facilitate intelligent scheduling decisions. Further, the vision capability repository hosts collections of implementations of different vision capabilities, and may also provide a request-response service that allows a device to request an available implementation of a particular vision capability.

[0167] In this manner, device-centric scheduling can be used to improve end-to-end (E2E) performance (e.g., latency and bandwidth efficiency) and scalability for visual fog computing.

[0168] FIG. **15** illustrates an example architecture **1500** for implementing device-centric scheduling in a visual computing system. In the illustrated embodiment, for example, visual computing architecture **1500** includes users **1502**, scheduling server **1504**, vision kernel repository **1506**, and various types of fog devices **1510**. A fog device **1510**, for example, can include any device ranging from the edge of a network to the cloud, inclusively. In the illustrated embodiment, for example, fog devices **1510** include cameras **1510***a*, gateways **1510***b*, and cloud servers **1510***c*.

[0169] In some embodiments, users **1502** may submit search queries for visual data captured by cameras **1510***a*. Moreover, in order to respond to those queries efficiently, scheduling server **1504** may schedule or distribute vision processing workloads across the various fog devices **1510**. In some embodiments, for example, scheduling server **1504** may perform intelligent scheduling decisions based on various criteria, such as the types of resources in the fog (e.g., the heterogeneous types of resources of the various fog devices **1510**), resource telemetry information (e.g., the availability of fog resources on a per-resource-type basis), and the implementations of vision capabilities that are available in the vision capability repository **1506**.

[0170] An example embodiment of the scheduling process, for example, is described below in connection with FIG. **16**.

[0171] FIG. **16** illustrates a flowchart **1600** for an example embodiment of device-centric scheduling in a visual computing system. In some embodiments, for example, flowchart **1600** may be implemented using visual computing architecture **1500** of FIG. **15**.

[0172] The flowchart may begin at block **1602** by collecting the available vision capability implementations. In some embodiments, for example, the scheduling server continuously synchronizes the collection of available implementations of vision capabilities from the vision capability repository.

[0173] The flowchart may then proceed to block **1604** to collect the resource telemetry of fog devices. In some embodiments, for example, the scheduling server may collect the resource availability of all fog devices on a per-resource-type basis. For example, the scheduling server may collect information regarding the resource availability of CPUs, GPUs, FPGAs, ASICs, and/or any other resource type across all fog devices.

[0174] In this manner, based on the available vision capability implementations collected at block **1602**, and the resource telemetry information collected at block **1604**, the scheduling server can subsequently schedule vision workloads based on the optimal task-to-device mapping in the visual fog paradigm.

[0175] For example, the flowchart may then proceed to block **1606** to determine whether a new vision workload has been received from a user. In some embodiments, for example, a user may submit a new visual query, which may require a new vision workload to be scheduled or distributed across the fog devices.

[0176] If it is determined at block **1606** that a new vision workload has NOT been received, the flowchart may then proceed back to block **1602** to continue synchronizing the available vision capability implementations and collecting resource telemetry information until a new vision workload is received.

[0177] If it is determined at block **1606** that a new vision workload has been received, the flowchart may then proceed to block **1608** to re-schedule all pending workloads. In some embodiments, for example, receiving a new vision workload for a user may trigger the scheduling server to re-schedule all pending workloads to ensure the collective workloads are distributed across the fog devices in the most efficient manner possible (e.g., based on the optimal task-to-device mapping).

[0178] In some embodiments, for example, scheduling may be performed based on various criteria, such as the types of fog resources that are available, telemetry information for those resources, and the vision capability implementations that are available for those fog resources.

[0179] In some embodiments, for example, a schedule that adheres to the constraints of multiple resource types can be determined using integer linear programming (ILP). Integer linear programming (ILP) is a mathematical optimization or feasibility technique for solving or optimizing a mathematical model represented by linear relationships. In particular, ILP can be used to optimize a linear objective function, subject to additional linear equality and linear inequality constraints. As an example, an ILP problem can be expressed as follows:

[0180] minimize: $c^T x$ (objective term)

[0181] subject to: $Ax \leq b$ (inequality constraint)

[0182] $Cx=d$ (equality constraint)

[0183] and: $x \in \{0, 1\}^K$ (binary constraint).

[0184] Moreover, this ILP model can be used to determine an optimal schedule f that satisfies a specified objective (e.g., total network utilization), while also adhering to other additional constraints (e.g., device resource constraints). In the above ILP model, for example, x presents the collection of possible schedules f, K is the length of x, the objective term presents a scheduling objective to be minimized (e.g., total network utilization), and the inequality/equality constraints present any additional constraints (e.g., device, resource, network, mapping, and/or privacy constraints). A device resource constraint, for example, can be presented as an inequality constraint of the ILP model. For example, in order to take into account constraints of multiple resource types, they can be expended into multiple inequalities in the form of Ax b in the ILP model above.

[0185] Accordingly, based on the scheduling decisions, the scheduling server assigns each fog device zero or more tasks. In some embodiments, for example, a task may be specified in a tuple of the form t=(p, r), where p denotes the vision capability and r denotes resource type (e.g., p=face detection, r=Movidius processor).

[0186] The flowchart may then proceed to block **1610** to determine if an updated workload schedule is available. For example, after a new vision workload is received and the pending workloads are re-scheduled, the scheduling server may have an updated or improved workload schedule that needs to be distributed to the fog devices. In some embodiments, however, the scheduling server may only update the workload schedule if the newly generated schedule is better or more efficient than the current workload schedule.

[0187] If it is determined at block **1610** that the workload schedule has NOT been updated, the flowchart may then proceed back to block **1602** to continue synchronizing the available vision capability implementations and collecting resource telemetry until the current workload schedule is eventually updated.

[0188] However, if it is determined at block **1610** that an updated workload schedule is available, the flowchart may then proceed to block **1612** to push the updated schedule to all fog devices.

[0189] The flowchart may then proceed to block **1614** to receive requests from fog devices for vision capability implementations. For example, each fog device may query the vision capability repository to request implementations of vision capabilities for the tasks assigned to the particular fog device. In some embodiments, for example, the request from a particular fog device may identify each of its assigned tasks t.

[0190] The flowchart may then proceed to block **1616** to identify the appropriate vision capability implementations for each fog device. In some embodiments, for example, the vision capability repository may be a dictionary of key-value pairs in the form of (task t, implementation i), where an implementation i can be distributed in various forms (e.g., a dynamic linking library in C/C++). Accordingly, based on the task(s) t specified in the request from a particular fog device, the vision capability repository identifies the corresponding implementation(s) i for that fog device. In some embodiments, for example, the vision capability repository identifies the optimal implementation of each vision capability requested by a fog device based on the available resources of that fog device.

[0191] The flowchart may then proceed to block **1618** to distribute the identified vision capability implementations to each fog device. In this manner, each fog device can then perform its assigned tasks using the appropriate vision capability implementations.

[0192] At this point, the flowchart may be complete. In some embodiments, however, the flowchart may restart and/or certain blocks may be repeated. For example, in some embodiments, the flowchart may restart at block **1602** to continue scheduling vision workloads.

[0193] FIG. **17** illustrates an example embodiment of a runtime processing pipeline **1700** for a visual fog architecture. In the illustrated embodiment, for example, a raw stream of visual data **1701** (e.g., video or images) captured by cameras or visual sensors in a visual fog architecture is provided as input to a stream ingress framework **1702**. The stream ingress framework **1702** decodes the raw stream of visual data **1701**, and a decoded stream **1703** is then provided as input to a distributed pre-processing framework **1704**. The distributed pre-processing framework **1704** then performs some preliminary processing using certain fog resources at the network edge (e.g., near the cameras or sensors that captured the visual data), such as data pre-processing, filtering, and/or aggregation. The resulting filtered stream **1705** may then be stored in data storage **1706** for subsequent use in responding to visual search queries and/or user-defined vision functions (UVFs) **1709** from end-users.

[0194] For example, end-users may subsequently submit visual search queries and/or user-defined vision functions (UVFs) **1709** associated with the visual data captured by the visual fog system. Accordingly, the UVFs **1709** are provided to a UVF compiler **1710**, which compiles the UVFs **1709** into a vision dataflow graph **1711** that can be used to execute the UVFs. For example, the vision dataflow graph **1711** is provided to a distributed UVF execution framework **1712**, which distributes or schedules workloads associated with the vision dataflow graph **1711** across the available fog nodes in the visual fog architecture.

[0195] After the workloads finish executing, the distributed UVF execution framework **1712** generates an output **1713** resulting from execution of the UVFs **1709**. For example, the output **1713** may include, or may be derived from, a filtered stream of visual data and/or metadata **1707** generated by execution of the UVFs **1709**. Moreover, in some embodiments, the resulting stream of visual data and/or metadata **1707** may then be stored in data storage **1706** for responding to subsequent visual search queries or UVFs.

[0196] Storage

[0197] As the volume of visual data generated in the real-world continues to grow, it is becoming increasingly common for visual data to be processed automatically by computers rather than manually reviewed by humans. Due to the increasing volume of visual data, however, data access has become a bottleneck in visual data processing, as existing visual data storage approaches suffer from various deficiencies.

[0198] To illustrate, image classification is a common visual data operation that uses a neural network to identify the contents of an image. For example, in machine learning, a convolutional neural network (CNN) is a type of feed-forward artificial neural network where the input is generally assumed to be an image. CNNs are commonly used for image classification, where the goal is to determine the contents of an image with some level of confidence. For example, a CNN is first trained for a specific classification task using a set of images whose object classes or features have been labeled, and the CNN can then be used to determine the probability of whether other images contain the respective object classes.

[0199] Visual data (e.g., images, video) must first be loaded from a storage system before it can be processed by a CNN. In the past, the data access latency has typically been less than the CNN vision processing latency, allowing the data access to be performed during the CNN processing. However, as hardware and software optimizations continue to improve the performance of CNN vision processing algorithms, the data access latency of existing solutions has become the bottleneck. Moreover, existing solutions typically store visual data in its original format rather than a format designed to aid with visual data processing, which further hinders performance.

[0200] Existing solutions are also unable to efficiently search visual data. For example, given a large data set (e.g., millions of images), the ability to efficiently identify a subset of relevant images using a query is important. The output of

a CNN used for image classification typically includes a vector of values corresponding to the probability of various objects existing in an image. However, existing solutions typically use this information for the task at hand and then discard it, requiring the processing to be repeated for subsequent use. For example, a CNN used to process an image with a dog and a cat may provide a probability for both, but if the goal was to find images with dogs, the information about cats is typically lost or discarded, thus preventing future use. In this manner, a subsequent search for images that contain cats would typically require the CNN to be run again on each image.

[0201] Accordingly, FIG. **18** illustrates an example embodiment of a visual data storage architecture **1800** designed to provide efficient access to visual data and eliminate the deficiencies of existing storage solutions used for visual data processing. In particular, storage architecture **1800** provides efficient metadata storage for searching visual data, as well as analysis-friendly formats for storing visual data.

[0202] In the illustrated embodiment, for example, storage architecture **1800** includes a request server **1802** for receiving visual search queries from a client API **1801**, a metadata database **1804**, a visual compute library **1806**, and a persistent data storage **1810**, as explained further below.

[0203] In some embodiments, for example, storage architecture **1800** may provide a unified API **1801** for visual data access (e.g., for both visual data and metadata). For example, visual data is commonly stored directly as files or in various types of databases (e.g., key-value, relational, and/or graph databases). Visual metadata is typically stored in databases, for example, while images and videos are typically stored as files. Moreover, different types of file systems and databases provide API functions in various programming and/or query languages in order to enable users to access and store data. Accordingly, in some embodiments, visual storage architecture **1800** may be implemented with a unified API (e.g., JSON-based) that supports multi-modal queries for retrieving any type of visual data from any storage source. In some embodiments, for example, the unified API could be used to retrieve and/or combine visual metadata and the original visual data from different storage locations. The unified API may also allow certain types of processing to be performed on visual data before it is returned to the requesting user. Further, the unified API may allow users to explicitly recognize visual entities such as images, feature vectors, and videos, and may simplify access to those visual entities based on their relationship with each other and with other entities associated with a particular vision application.

[0204] Moreover, in some embodiments, a multi-tier lazy data storage approach may be used to store visual data more efficiently (e.g., using long- or short-term storage in different portions of the distributed edge-to-cloud network). For example, multiple storage tiers may be used to store visual data in different locations and for varying amounts of time based on the type or importance of the visual data. In some embodiments, for example, video cameras may store all video captured within the past day, gateways may store video with motion activities within the past week, and the cloud may store video associated with certain significant events within the past year.

[0205] Similarly, intelligent placement and aging of visual data across the storage tiers may further improve the data storage efficiency (e.g., determining where to store the visual data within the distributed edge-to-cloud system, when the data should be moved from hot to warm to cold storage, and so forth). For example, visual data and metadata can be distinguished and segregated based on data access patterns. Moreover, analysis friendly storage formats can be used to enable data to be read faster when needed for vision processing. These various data formats may be used to form the hot, warm, and cold tiers of data that can be mapped to various heterogeneous memory and storage technologies, based on the intended use and lifetime of the data. For example, storage tiers can be used to represent hot, cold, and optionally warm data. Hot data is accessed frequently; warm data is accessed occasionally; and cold data is accessed rarely (if ever). Accordingly, cold data may be stored on slower hardware since low access latency for retrieval of the data is less important. In this manner, intelligent decisions can be used to determine when and which portions of visual data should remain in the hot tiers and when it should be migrated to colder tiers, and which storage format should be used. For example, regions of interest may remain in hot storage in the analysis friendly format much longer than the entire image/video.

[0206] Metadata database **1804** is used to store metadata in a manner that facilitates efficient searches of visual data. For example, when performing image classification using a CNN, the resulting image-object relationships or probabilities can be stored as metadata, and the metadata can be used for subsequent searches of the images, thus eliminating the need to repeatedly process the images for each search. For example, FIG. **19** illustrates an example of a vision processing pipeline **1900** that leverages metadata for searching visual data. In the illustrated example, a stream of incoming visual data is received from a network or file system at block **1902**, vision processing is performed on the visual data to derive metadata (e.g., using a CNN) at block **1904**, the metadata is stored at block **1906**, search queries for relevant visual data are received at block **1908**, and the search queries are then satisfied using either the metadata obtained at block **1906** or additional vision processing performed at block **1904**.

[0207] In some embodiments, storage architecture **1800** may store visual metadata as a property graph to identify relationships between visual data, such as images that contain the same object or person, images taken in the same location, and so forth. For example, FIGS. **20** and **21** illustrate examples of representing visual metadata using a property graph. In this manner, visual metadata can be easily searched to identify these relationships, thus enabling flexible search queries such as "find all images taken at location Y that contain person A."

[0208] Moreover, in some embodiments, metadata database **1804** of storage architecture **1800** may be implemented as a persistent memory graph database (PMGD) to enable visual metadata to be searched more efficiently. For example, using persistent memory (PM) technology, a graph database containing the visual metadata can be stored both in-memory and persistently. In this manner, a persistent memory graph database (PMGD) can be designed to leverage a memory hierarchy with data structures and transactional semantics that work with the PM caching architecture, reduce write requests (addressing PM's lower write bandwidth compared to DRAM), and reduce the number of

flushes and memory commits. This approach enables a graph database of visual metadata to be searched efficiently to identify relevant visual data.

[0209] Further, feature vector storage optimizations may be used to achieve fast searching of visual metadata. For example, feature vectors can be generated by various vision algorithms to identify regions or features of interest in visual data (e.g., faces, people, objects), and they are typically represented as vectors of n-dimensional floating-point values. Finding the nearest neighbor for a given feature vector is a common operation that is computationally expensive, especially at the cloud scale due to billions of potential feature vectors (e.g., a feature vector for each interesting region of each image or video frame). Accordingly, in some embodiments, feature vectors may be represented and stored as visual metadata using an efficient format. For example, visual metadata may be stored using an analysis-friendly array format that indicates where the feature vectors reside, and an index may be built on interesting dimensions within the metadata storage to narrow the search space.

[0210] Storage architecture **1800** also includes a separate data storage **1810** for storing the visual data itself, such as images or videos. Segregating the metadata and visual data in this manner enables each type of data to be mapped to the most suitable hardware in a heterogeneous system, thus providing flexibility for the request server **1802** to identify the most efficient way to handle a visual data request.

[0211] Moreover, storage architecture **1800** is also capable of storing visual data on data storage **1810** using an analytic image format designed to aid in visual processing. In the illustrated embodiment, for example, visual compute library (VCL) **1806** of storage architecture **1800** is designed to handle processing on analytic image formats **1807** in addition to traditional formats **1808**. For example, visual compute library **1806** can implement an analytic image format **1807** using an array-based data management system such as TileDB, as described further with respect to FIG. **22**. The analytic image format **1807** provides fast access to image data and regions of interest within an image. Moreover, since the analytic image format **1807** stores image data as an array, the analytic image format **1807** enables visual compute library **1806** to perform computations directly on the array of image data. Visual compute library **1806** can also convert images between the analytic image format **1807** and traditional image formats **1808** (e.g., JPEG and PNG). Similarly, videos may be stored using a machine-friendly video format designed to facilitate machine-based analysis. For example, videos are typically encoded, compressed, and stored under the assumption that they will be consumed by humans. That assumption is often leveraged for video encoding by eliminating information that human eyes and brains cannot process. Videos intended for machine-based processing, however, may benefit from alternative storage methods designed to speed up the time required to retrieve full images or regions of interest within a video or video frame, and even enhance the accuracy of machine-learning video processing mechanisms.

[0212] FIG. **22** illustrates an example embodiment of an analytic image format **2200** designed to aid in visual data processing. In some embodiments, for example, storage architecture **1800** may use analytic image format **2200** to store images in a format that facilitates visual data processing and analysis.

[0213] Deep learning neural networks, such as CNNs, are frequently used for image processing, including object/edge detection, segmentation, and classification, among other examples. Images are typically read from disk during both training and inferencing, for example, using background threads to pre-fetch images from disk and overlap the disk fetch and decode times with the other compute threads. However, compute cycles may still be wasted reading the images from disk and decompressing/decoding the images to prepare them for processing, thus reducing the overall throughput (e.g., images/second) of an image processing system.

[0214] Moreover, traditional lossy image formats (e.g., JPEG) are designed to compress image data by discarding high-frequency information that is not perceptible by humans. While the discarded information may be meaningless to humans, however, it can improve the accuracy and performance of deep learning neural networks used for image processing.

[0215] For example, images can be compressed either in a lossless or lossy manner. Lossless image compression preserves all the information in the image, while lossy compression takes advantage of visual perception and statistical properties to achieve better compression rates, but results in some data being lost. The JPEG compression algorithm is a commonly used lossy algorithm that is often used for images on the web. The JPEG algorithm is based on discrete cosine transforms (DCT), and discards high-frequency details that are not perceptible to the human eye, which results in much smaller image file sizes. However, in cases where exact image reproduction is required, or when the image will be edited multiple times, lossless compression is preferred. For example, PNG is an image file format that supports lossless compression using a bitmap image. With PNG, images are transformed using a filter type on a per-line basis, and then compressed using the DEFLATE algorithm. There are numerous other image formats with similar technologies behind them that are suitable for different applications and use cases. While a traditional lossless image format (e.g., PNG) could be used to retain all image data for image processing purposes, that comes at the cost of a lower compression rate.

[0216] Further, images stored using traditional formats (e.g., JPEG and PNG) must be converted into an internal array format before any processing can begin. For example, before any operations can be performed on images stored using traditional formats, the entire image file must be read from disk and decoded into an internal array format. In analytics, however, operations such as resizing and cropping are often performed before any sort of learning or understanding happens, thus rendering traditional image formats inefficient for image processing and analytics.

[0217] Accordingly, traditional image formats (e.g., JPEG and PNG) are designed for human consumption, and performing operations on them is often time-consuming and inefficient. Moreover, lossy image formats (e.g., JPEG) discard information that may be useful in machine learning, and thus are not well-suited for image processing. Moreover, while existing database management systems could be used to store images, they are not designed for image data and thus do not store image data efficiently.

[0218] The analytic image format **2200** of FIG. **22** is designed to aid in image processing and alleviate the deficiencies of existing image formats. For example, image

format **2200** is implemented using an array-based data storage format that is lossless and eliminates the expensive decoding process that is required for processing traditional image formats. In some embodiments, for example, analytic image format **2200** could be implemented using an array-based data storage manager such as TileDB. TileDB is a data management system designed for efficiently managing large volumes of scientific data represented using arrays. While TileDB is not specific to images, it is designed to provide fast access to array-based data. Accordingly, in some embodiments, image format **2200** can be implemented using TileDB to achieve the performance boost of TileDB for image processing purposes.

[0219] In some embodiments, for example, analytic image format **2200** can be implemented by defining how the pixel data of an image is stored and accessed in an array-based format (e.g., using an array-based data storage manager such as TileDB). In this manner, image format **2200** enables efficiency in processing large images, which reduces the overall time for image analytics. As visual understanding algorithms get faster and the hardware to perform the algorithms gets better, the time to retrieve and process the images is becoming more and more significant. However, by using analytic image format **2200**, storage and retrieval of images does not become a bottleneck in the visual processing pipeline.

[0220] For example, analytic image format **2200** allows an image to be stored as a lossless compressed array of pixel values. Accordingly, when image data is needed for processing, the image data does not need to be decoded before being processed, as required for traditional image formats. This improves the speed at which data is retrieved and made usable, yet still provides some level of compression. While this approach requires images to be written to the analytic image format **2200** prior to training or inference, the additional write overhead is minimal.

[0221] Moreover, because TileDB outperforms many array database managers for both sparse and dense data access, it is an ideal choice for implementing analytic image format **2200**. In other embodiments, however, analytic image format **2200** can be implemented using any other type of array-based data manager or data format. The use of a fast, enhanced array storage system such as TileDB enables image format **2200** to eliminate slow reads of images from disk, and remove the in-loop conversion of traditional image formats to arrays.

[0222] Image format **2200** is also beneficial in applications where subarray accesses are common, such as accessing regions of interest in an image. For example, an array data manager such as TileDB can be used to improve the speed of common operations that are needed for image analytics, such as resize and crop, by enabling fast subarray accesses.

[0223] FIG. **22** illustrates the process of converting an image into an analytic image format **2200** using an array-based data manager such as TileDB. In the illustrated example, the original image is first received **2202** and is then divided into a plurality of tiles **2204** using an optimal tile size, and the tiles are then compressed and written to memory on a per-tile basis **2206** using an array-based storage format.

[0224] In some embodiments, the optimal tile size for analytic operations can be dynamically determined for each image. For example, in order to determine the optimal tile size for a particular image, a random portion of the image may be selected and then processed using different tile sizes and compression algorithms in order to determine the ideal tile size and compression for that image. Moreover, since image processing operations are often postponed until the data is actually needed, there is a period of time available to carry out the experimentation without impacting performance.

[0225] An image that does not fit perfectly into tiles of the selected tile size will have partially empty tiles that are padded with empty characters, as depicted in FIG. **22**. In this manner, the original size of the image may be stored as metadata (e.g., height, width, and number of channels), and when the image is subsequently read from storage, the metadata can be checked to determine the actual dimensions of the image to avoid reading the empty characters or padding.

[0226] For high-resolution images, image format **2200** improves the speed of common operations such as reading and writing, as well as the speed of operations used in image analytics, such as cropping and resizing. For example, storing images using image format **2200** improves read performance, as the images are compressed but not encoded, and thus do not need to be decoded when they are read from the file system. In addition, image format **2200** enables fast access to subarrays of image pixels, making cropping a simple matter of reading a particular subarray rather than reading the entire image and then cropping it to the appropriate size.

[0227] For example, FIG. **23** illustrates a graph **2300** comparing the performance of analytic image format **2200** from FIG. **22** with the PNG image format, which is a traditional lossless image format. As shown by FIG. **23**, the analytic image format provides better performance than PNG for writes, reads, crops, and resizes. The largest improvement is seen in cropping, as the analytic image format allows only the pertinent information to be read from the file, rather than reading the entire image file and then cropping to the desired size. Accordingly, the performance improvement for common data access and analytic operations demonstrates that analytic image format **2200** is highly beneficial for image processing purposes.

[0228] FIG. **50** illustrates an example write processing flow **5000** for traditional and analytic image formats. In the illustrated processing flow **5000**, for example, raw pixel data **5002** can be written to disk **5010** using either a traditional image format or an analytic image format. The top path of processing flow **5000** illustrates the flow for writing traditional image formats (e.g., PNG), while the bottom path illustrates the flow for writing analytic image formats.

[0229] With respect to traditional image formats, for example, raw pixel data **5002** is encoded **5004**, compressed **5006**, and then stored **5010**. With respect to analytic image formats, however, raw pixel data **5002** is compressed **5008** and then stored **5010**, but the encoding step is omitted. While the resulting analytic image format may result in a larger file size on disk, the latency of data access operations (e.g., writes) and other image operations may be reduced.

[0230] Moreover, the read processing flow for traditional and analytic image formats may be implemented as the reverse of the write processing flow **5000**. For example, with respect to traditional image formats, the encoded/compressed data is read from disk, decompressed, and then decoded into the original image. With respect to analytic image formats, the compressed data is read from disk and

then decompressed into the original image, but the decoding step is omitted since the encoding step was omitted during the write processing flow **5000**.

[0231] TABLE 1 illustrates an example analytic image format schema. In some embodiments, for example, the analytic image format schema of TABLE 1 could be implemented using an array-based database manager (e.g., TileDB) to store images as dense arrays.

TABLE 1

example analytic image format

PARAMETER	TYPE	EXAMPLE VALUE
cell order	fixed	row major
tile order	fixed	row major
number of dimensions	fixed	2
dimension names	fixed	"height", "width"
number of attributes	fixed	1
compression	fixed	LZ4
array height	variable	3534
array width	variable	5299
domain	variable	[0, 3533, 0, 5298]
tile height	variable	589
tile width	variable	757

[0232] The schema of TABLE 1 specifies parameters about the array that can be used to arrange the image data. Moreover, some parameters of the analytic image format are fixed, while others are determined on a per-image basis. For example, images have only two dimensions, a height and a width, thus fixing the number of dimensions as well as the names of the dimensions. The number of attributes is set to one, which means each cell holds the blue, green, and red (BGR) values for the corresponding pixel. All three values are generally read together, as a pixel is defined by all three values. In other embodiments, however, the color values may be stored separately. The intra-tile and array-level tile ordering is fixed to be row major. Row major order means that data is written and read from left to right in rows within a tile, and tiles are written and read in the same manner. This information allows the array database to efficiently perform subarray reads.

[0233] The dimensions and domain of the array depend on the resolution of the original image and therefore are calculated dynamically on a per-image basis. Since images often do not have an evenly divisible number of pixels in one or both dimensions, this occasionally results in the dimensions of an array not matching the original resolution of the image. This is reflected in TABLE 1, where the array height is one pixel larger than the image height. To make up the difference between an image dimension and an array domain, the image is padded with empty characters. An example of this can be seen in FIG. **22**, where the white space within certain tiles corresponds to empty characters. In the actual array, the size of the array domain is increased by a single pixel when needed. The original size of the image (height, width, and number of channels) is stored as metadata by default. When an image in the analytic format is read, the metadata is read first in order to determine the dimensions of the image, thus avoiding reading the empty characters.

[0234] Tile extents depend on the array dimensions and are calculated once the array dimensions are known. All tiles have the same height and width. The optimal number of tiles may vary based on image content and resolution, and thus in some embodiments, the optimal number of tiles may be determined on a per-image basis. For example, in order to determine the best tile size, a portion of the image may be randomly selected and tested using different tile sizes and compression algorithms to determine the best combination for that image. Since all operations are postponed until the data is actually needed, there is a period of time to carry out the experimentation that does not affect the performance. In other embodiments, however, a predefined minimum number of tiles per dimension (e.g., 4 tiles per dimension) may be used as a basis to determine tile height and width.

[0235] The compression algorithm used to compress the analytic image data has a fixed default (e.g., the LZ4 compression algorithm), but other compression algorithms can be set manually.

[0236] FIG. **51** illustrates an example embodiment of a visual compute library (VCL) **5100** for traditional and analytic image formats. For example, VCL **5100** provides an interface through which a user can interact with the analytic image format as well as traditional image formats.

[0237] When a user creates an analytic image using VCL **5100**, the analytic image schema is automatically set using the parameters described above in TABLE 1. VCL **5100** then creates a layer of abstraction with function calls of TileDB **5102** (e.g., the array-database manager used in the illustrated embodiment) combined with specialized transformation operations to provide an interface to the analytic image. VCL **5100** also extends the abstraction layer to OpenCV **5104**, providing support for PNG and JPEG image formats. VCL **5100** uses OpenCV **5104** to perform both I/O and transformation operations on images that are stored in either PNG or JPEG format. For images stored in the analytic format, VCL **5100** handles the transformation operations and uses TileDB **5102** for I/O operations.

[0238] To initially store an image in the analytic format, the raw pixel data of an image is passed to VCL **5100** in some manner (e.g., as a path to a PNG or JPEG file stored on disk, an OpenCV matrix, a buffer of encoded pixel data, a buffer of raw pixel data, and so forth). This data is converted to a raw pixel buffer in order to write to the analytic format. Since the TileDB array schema for images has already been set at this point (e.g., using the parameters of TABLE 1), the TileDB functions can be used to write the data to disk.

[0239] Reading an image in the analytic format requires the metadata to be read first to determine the original image resolution. This ensures that only image data is read and that empty characters are ignored. The raw analytic-format or TileDB data is read into a buffer, keeping the data in the order in which it was written, which is referred to as "tile order" (e.g., as illustrated in FIG. **52**). This is because if the data never needs to be returned to the user (e.g., if the user just wants to manipulate it and write it out again), it is faster to use the tile order buffer. In cases where the data is to be returned to the user, however, the buffer is re-ordered into image order, which results in a buffer that has each row of the image sequentially (e.g., as illustrated in FIG. **52**). Image order, for example, is typically expected by other programs such as OpenCV **5104**.

[0240] Crop, another frequently used operation in image processing, is used to retrieve a region of interest within an image for processing. Rather than reading the entire image and then selecting a sub-region (as is required for traditional image formats), the analytic or TileDB crop function uses

the crop parameters to specify a subarray of the analytic image data. The subarray is then the only portion of the image that is read.

[0241] Resize, another frequently used operation in image processing, is used to resize the dimensions of an image (e.g., to either a smaller or larger size). The TileDB resize occurs after the image has been read, but while the data is still in tile order. VCL **5100** implements a version of resize for TileDB that uses a bilinear interpolation, following the OpenCV default. For example, in a linear interpolation, a new value is calculated based on two points; bilinear interpolation does this in two different directions and then takes a linear interpolation of the results. These points are identified by (row, column) in the original image. Given the data is in tile order, it is necessary to identify which tile each point is part of in order to locate the value of that point in the buffer. The resulting resized image buffer is in image order, although other approaches may be used to keep it in tile order.

[0242] Compression/Compressive Learning

[0243] The performance of large-scale visual processing systems can be improved using efficient compression algorithms and techniques for storing and processing visual data. The compression approaches of existing visual processing solutions, however, suffer from various deficiencies. For example, existing solutions require visual data to be fully decompressed before any processing can be performed (e.g., using deep learning neural networks). Moreover, existing solutions typically compress and store images individually, thus failing to leverage the potential compressive benefits of collections of similar or related images with redundant visual data.

[0244] Accordingly, this disclosure presents various embodiments for compressing and processing visual data more efficiently. In some embodiments, for example, neural networks can be designed to operate on compressed visual data directly, thus eliminating the need to decompress visual data before it can be processed. Moreover, context-aware compression techniques can be used to compress visual data and/or visual metadata more efficiently. For example, context-aware compression can be used to compress distinct instances of redundant visual data more efficiently, such as a group of images taken close in time, at the same location, and/or of the same object. Similarly, context-aware compression can be used to compress visual metadata more efficiently (e.g., using a context-aware lossless compression codec). In some embodiments, for example, visual metadata could be compressed by pre-training a convolutional neural network (CNN) to classify visual metadata, replacing long strings of visual metadata with shorter symbols (e.g., predefined human codes), performing multi-scale de-duplication on the visual metadata, and finally compressing the resulting visual metadata using a compression algorithm (e.g., the LZ77 lossless compression algorithm or another similar alternative).

[0245] FIGS. **24**A-C illustrate an example embodiment of a multi-domain cascade convolutional neural network (CNN) **2400**.

[0246] In distributed visual analytics systems, image and video is often compressed before transmission (e.g., from the pixel domain to a compressed domain), and subsequently decompressed after transmission (e.g., back to the pixel domain) before any processing can be performed, such as deep learning using neural networks. As an example, image and video captured by edge devices may be compressed and transmitted to the cloud, and then decompressed by the cloud before any further processing begins.

[0247] This approach suffers from various disadvantages. First, extra computation is required to fully decompress the visual data before it can be processed, thus significantly increasing the total processing time (e.g., by up to 100% in some cases). For example, before processing can be performed, the visual data must be fully decompressed back to the pixel domain using hardware or software decoding. Accordingly, given that not all processors include built-in video decompression accelerators, decompression may incur an additional cost for video analytics.

[0248] Next, extra bandwidth is required to transmit the decompressed data between separate processing components (e.g., between a decompression engine and an analysis engine), thus significantly increasing bandwidth usage (e.g., by up to 20 times in some cases).

[0249] Moreover, the requirement to fully decompress visual data prior to processing precludes the ability to leverage a fully distributed neural network in the edge-to-cloud sense. For example, the use of distributed analytics to process visual data exclusively in the pixel domain requires the visual data to be analyzed at multiple scales.

[0250] Further, relying on the cloud to perform processing on visual data captured by edge devices often results in wasted transmission bandwidth, as many images or videos transmitted from the edge to the cloud may not contain any objects or features of interest. In many cases, for example, it could be possible to perform object detection and classification closer to the network edge (e.g., near the sensors that capture the visual data) using lower complexity analytics algorithms, potentially saving the transmission cost of insignificant or unimportant data.

[0251] Accordingly, FIGS. **24**A-C illustrate an example embodiment of a multi-domain cascade CNN **2400** that can be used to process visual data in the compressed and pixel domains, thus eliminating the requirement to decompress visual data before it can be processed. In this manner, multi-domain cascade CNN **2400** can be used to perform distributed visual analytics in a visual fog system using compressed domain data as input.

[0252] In some embodiments, for example, multi-domain cascade CNN **2400** may be a cascaded CNN that includes multiple decision stages. For example, in a first or early decision stage, a subset of the compressed domain visual data or features may be used (e.g., motion vectors) to attempt to generate an early decision. If the visual data cannot be detected or classified in the early stage, additional compressed domain data (e.g., motion prediction residuals) may be provided as input to a subsequent or late decision stage. Finally, for improved accuracy and/or in the event the late decision stage is unsuccessful, the visual data may be fully decompressed and a final decision stage may be performed using the decompressed visual data.

[0253] In the illustrated embodiment, for example, CNN **2400** includes an early decision stage (illustrated in FIG. **24**A), a late decision stage (illustrated in FIG. **24**B), and a final decision stage (illustrated in FIG. **24**C). Moreover, CNN **2400** is designed to process compressed visual data **2402** as input (e.g., video sequence data compressed with a motion-compensated predictive coding scheme such as H.264).

[0254] In some embodiments, for example, compressed visual data **2402** provided as input to CNN **2400** may first be partially decoded to separate and extract different syntax elements (e.g., motion vectors, macroblock (MB) coding modes, quantized prediction residuals), thus producing a subset of partial compression data **2404**.

[0255] As shown in FIG. **24**A, in the early decision stage, the partial compression data **2404** (e.g., motion vectors) is provided as input to a first stage CNN **2405***a* to attempt to identify an early decision **2406**. In some embodiments, the CNN processing may then terminate if an early decision can be made. For example, in some embodiments, the early decision stage may be performed by a fog or edge node near the sensor that captured the visual data. Accordingly, if an early decision can be made, it may be unnecessary to transmit additional visual data to another node (e.g., in the cloud) for a subsequent processing stage, thus saving bandwidth and/or resources (e.g., energy) that would otherwise be required for the later stage. For example, assuming the goal is to detect moving pedestrians using traffic cameras, if there is no motion detected, there likely are no moving objects. Accordingly, an early decision can be made, and any further transmission or processing of the visual data can be aborted. In other embodiments, however, the subsequent CNN processing stages of CNN **2400** may still be performed even if an early decision can be made. Moreover, the complexity of the first stage CNN **2405***a* may vary based on different use cases, resource availability, and so forth.

[0256] If the early decision stage is unable to detect or classify the partial compression data **2404** using the first stage CNN **2405***a*, CNN **2400** may proceed to a late decision stage, as shown in FIG. **24**B. In the late decision stage of FIG. **24**B, for example, additional compression data **2410** (e.g., motion prediction residuals) is evaluated using a second stage CNN **2405***b* to attempt to determine a late decision **2408**.

[0257] Finally, for improved accuracy and/or in the event the late decision stage is unsuccessful (e.g., the late decision stage is unable to detect or classify the additional compression data **2410** using the second stage CNN **2405***b*), CNN **2400** may proceed to a final decision stage, as shown in FIG. **24**C. In the final decision stage of FIG. **24**C, for example, the compressed visual data **2402** may be fully decompressed using a decompression engine **2412**, and the decompressed visual data **2414** (e.g., pixel domain data) may then be evaluated using a final stage CNN **2405***c* to determine a final decision **2416**.

[0258] Accordingly, the collective stages of multi-domain cascade CNN **2400** are depicted in FIG. **24**C, where an early stage is used to generate an early decision based on an initial subset of compressed domain data, and later stages are used to generate re-fined or final decisions based on additional compressed domain data and eventually pixel domain data.

[0259] The described embodiments of multi-domain cascade CNN **2400** provide numerous advantages. First, visual data (e.g., images or video) does not need to be fully decompressed before its contents can be analyzed using deep learning neural networks, thus reducing memory usage and computation typically required for decoding or decompressing the visual data. Next, the cascading approach of CNN **2400** avoids the need to transmit certain compressed data to the cloud, such as when an early decision can be reached by an edge or fog node, thus improving bandwidth usage. Finally, a large portion of the overall analysis often occurs in the early decision stage, which typically involves a simplified CNN or machine learning model, thus reducing the overall computational complexity.

[0260] FIGS. **25-31** illustrate the use of butterfly operations to implement a multi-domain convolutional neural network (CNN) that is capable of processing both raw and compressed visual data.

[0261] As discussed above, many visual analytics systems require visual data to be fully decompressed before any visual processing can be performed (e.g., using deep learning neural networks), which is an approach that suffers from various inefficiencies, including higher processing latency, additional transmission bandwidth, and so forth. Accordingly, this disclosure presents various embodiments of a deep learning neural network that is capable of analyzing compressed visual data directly. In particular, the described embodiments present a multi-domain CNN that uses butterfly operations to enable visual data processing in either the pixel domain or the compressed domain.

[0262] To illustrate, existing deep learning CNNs (e.g., inception or ResNet CNN models) typically repeat an inner module multiple times, and the inner module aggregates the results from multiple convolution layers and/or the original input at the end (analogous to a bottleneck). For example, FIGS. **25**A-B illustrate a traditional 27-layer inception model CNN **2500**, and FIGS. **26** and **27** illustrate example inner modules **2600** and **2700** for an inception model CNN. In particular, FIG. **26** illustrates an inner module **2600** implemented without dimension reduction, while FIG. **27** illustrates an inner module **2700** implemented with dimension reduction. These CNN implementations are designed to process visual data in the pixel domain (e.g., raw or uncompressed visual data).

[0263] FIGS. **28** and **29**, however, illustrate example CNN inner modules **2800** and **2900** that use butterfly operations to enable multi-domain visual data processing in either the pixel domain or the compressed domain. Butterfly operations, for example, are operations that can be used to transform compressed domain data (e.g., DCT domain data) back to the pixel domain. Accordingly, by incorporating butterfly layers into a CNN, the CNN can be provided with compressed visual data as its original input, and as the compressed data is processed by the successive CNN layers, the compressed data is at least partially transformed or decompressed back to the pixel domain using the butterfly layers in the CNN.

[0264] FIG. **28** illustrates an inner CNN module **2800** implemented without dimension reduction, while FIG. **29** illustrates an inner CNN module **2900** implemented with dimension reduction. Moreover, as shown in these examples, additional butterfly layers or filters are added in parallel to the regular convolution layers. In some embodiments, for example, 2×2 and/or 4×4 butterfly operations can be added in parallel to the regular convolution and pooling layers. For example, in some embodiments, the butterfly operations could be implemented similar to the example butterfly operation illustrated in FIGS. **31**A-B.

[0265] With respect to inner module **2800** of FIG. **28**, for example, butterfly layers **2830***a,b* are added in parallel to convolution layers **2810***a-c* and pooling layer **2820**, and the butterfly layers **2830** include vertical N-point butterfly operations **2830***a* and horizontal N-point butterfly operations **2830***b*. For example, in some embodiments, the butterfly operations may be performed for both the vertical

pixels and the horizontal pixels. Similarly, with respect to inner module **2900** of FIG. **29**, butterfly layers **2930***a,b* are added in parallel to convolution layers **2910***a-e* and pooling layers **2920***a-b*, and the butterfly layers **2930** include vertical N-point butterfly operations **2930***a* and horizontal N-point butterfly operations **2930***b*.

[0266] Note that this approach, however, does not require multiple butterfly layers to be stacked within a single inner module, as the CNN does not have to perform a complete inverse DCT. For example, the goal of multiple convolution layers is to extract/transform the input data to a feature space where the fully connected layers can easily separate different clusters. Accordingly, the butterfly layers do not have to perform a complete inverse DCT, and instead, they can simply be designed to aid in extracting and transforming the input data into the feature space. In this manner, a complete or entire stack or organized butterfly layers does not need to be included in the CNN.

[0267] Moreover, the weights of each butterfly can be adjusted during the training phase, and thus the decision of whether to use the butterfly layers and/or how much to rely on them will be adjusted automatically.

[0268] FIG. **30** illustrates an alternative embodiment of a multi-domain CNN **3000** with butterfly layers **3002** and normal layers **3004** arranged sequentially rather than in parallel.

[0269] FIGS. **31**A-B illustrate an example of a one-dimensional (1D) N-point butterfly operation. In particular, the illustrated example is a 4-point butterfly operation, meaning the butterfly operation is performed using four data points **3110***a-d*. In other embodiments, however, butterfly operations may be implemented using any number of data points. Moreover, in some embodiments, data points **3110***a-d* may represent compressed pixel data, such as DCT coefficients.

[0270] In some embodiments, the butterfly operation may be performed in multiple stages. In each stage, for example, the butterfly operation may generate two outputs or channels using separate addition and subtraction operations (e.g., by computing the sum of two points over a large distance and the difference of two points over a large distance). For example, during a particular stage, the 1^{st} and 4^{th} points may be added together to compute their sum (1^{st} point+4^{th} point), and also subtracted to compute their difference (1^{st} point−4^{th} point). The points may then be shifted up cyclically and the process may be repeated for the next stage. For example, after each stage, the 4^{th} point becomes the 3^{rd} point, the 3^{rd} point becomes the 2^{nd} point, the 2^{nd} point becomes the 1^{st} point, and the 1^{st} point becomes the 4^{th} point. After the points are shifted, the next stage of the butterfly operation is performed by repeating the addition and subtraction on the 1^{st} on 4^{th} points (e.g., using the new ordering of points).

[0271] In FIGS. **31**A-B, for example, the addition and subtraction operations for the first stage of a butterfly operation are shown. In particular, FIG. **31**A illustrates the addition operation, and FIG. **31**B illustrates the subtraction operation. In FIG. **31**A, for example, the 1^{st} point (**3110***a*) and the 4^{th} point (**3110***d*) are added together to compute a new point (**3120***a*) that represents their sum. Similarly, in FIG. **31**B, the 4^{th} point (**3110***d*) is subtracted from the 1^{st} point (**3110***a*) to compute a new point (**3130***d*) that represents their difference. The points are then shifted in the manner described above to perform the subsequent stages of the butterfly operation.

[0272] Accordingly, the butterfly operations can be incorporated into a CNN in this manner in order to enable processing of visual data in both the pixel domain and compressed domain (e.g., DCT domain), thus eliminating the requirement of fully decompressing visual data before analyzing its contents using a deep learning neural network. For example, rather than explicitly performing an inverse DCT transform to fully decompress visual data before processing it using a CNN, the CNN can instead be implemented using butterfly layers to inherently incorporate decompression functionality into the CNN, thus enabling the CNN to be provided with compressed data as input.

[0273] FIGS. **32** and **33** illustrate an example embodiment of a three-dimensional (3D) CNN **3200** that is capable of processing compressed visual data. In some embodiments, for example, 3D CNN **3200** could be used in the implementation of, or in conjunction with, the compression-based CNN embodiments described throughout this disclosure (e.g., the CNNs of FIGS. **24** and **28-31**).

[0274] Many visual analytics systems require visual data to be decompressed before any processing can be performed, such as processing by a deep learning neural network. To illustrate, FIG. **34** illustrates an example of a pixel-domain CNN **3400**, and FIG. **35** illustrates an example of an associated pixel-domain visual analytics pipeline **3500**. In the illustrated example, pixel-domain CNN **3400** performs object detection and classification for visual analytics using data in the pixel or image domain (e.g., using decompressed visual data). For example, the convolutional kernels in the early layers of the CNN implement two-dimensional (2D) convolutions on the image data, and multiple layers of convolutions, pooling, and rectified linear unit (ReLU) operations are repeated in order to successively extract combinations of features from the earlier layers. Moreover, because CNN **3400** operates on pixel-domain data, compressed visual data must be fully decompressed before it can be processed by CNN **3400**. For example, as shown by visual analytics pipeline **3500** of FIG. **35**, the original pixel domain data **3502** is first compressed by a video encoder **3510** (e.g., prior to transmission over a network), and the compressed data **3504** is subsequently decompressed by a video decoder **3520** before performing video analytics **3540** (e.g., using a CNN).

[0275] In the illustrated embodiment of FIGS. **32** and **33**, however, 3D CNN **3200** processes compressed visual data directly using a 3D format designed to improve processing efficiency. For example, the input image may be transformed into the DCT domain and reshaped into a 3D format in order to separate the DCT transform coefficients into different channels. In this manner, the reshaped DCT transform data is arranged in a manner that provides better correlation between the spatial and transform domain coefficients. The reshaped DCT transform data can then be processed directly by a CNN (e.g., using 3D convolutions to perform feature extraction), which ultimately enables the CNN to be trained faster. For example, by eliminating the decompression step required by existing approaches, processing efficiency is improved, particularly for computing environments that do not include built-in hardware video decompression accelerators.

[0276] In some embodiments, for example, 3D CNN **3200** may be designed to operate directly on compressed visual data (e.g., video frames) represented in the DCT domain using a 3D matrix. For example, in some embodiments, the

DCT block indices may be represented by the x and y dimensions of the 3D matrix, while the DCT transform magnitude vectors may be organized along the z dimension. In this manner, the convolutional kernels in the first layer of the new CNN architecture can be implemented using 3D filters designed to better capture the spatial and frequency domain correlations and features of the compressed data, thus improving the performance of the CNN operation in the DCT domain.

[0277] The majority of common video and image encoding schemes use discrete cosine transforms (DCT) to convert spatial pixel intensities to frequency domain representations. The illustrated embodiment is based on the observation that once image data is split into 4×4 pixel blocks and passed through a transform such as DCT, the transformed data has different correlation properties than the original data. For example, with respect to a DCT transform, the DC coefficients of adjacent blocks are often strongly correlated, while the corresponding higher frequency AC coefficients of adjacent blocks may be similarly correlated.

[0278] Accordingly, FIG. **32** illustrates an approach for transforming a 2D image into a 3D matrix of DCT data, which is arranged in a manner that allows the DCT data to be processed more efficiently by a CNN. In the illustrated example, an input image of size N×N (reference numeral **3210**) is first broken up into 4×4 pixel blocks (example reference numeral **3212**), and each 4×4 pixel block is passed through a DCT transform. The resulting DCT transform domain data (reference numeral **3220**) is then stored in a 3D matrix, where the x and y dimensions correspond to the spatial block indices and the z dimension contains vectors of DCT coefficients (reference numeral **3222**), which include 16 coefficients per vector. Accordingly, the resulting transform domain data (reference label **3220**) has dimensions of size K×K×16, where K=N/4.

[0279] Next, as shown in FIG. **33**, the transform domain data represented using the 3D matrix (reference label **3220**) is input into the CNN (reference label **3200**), which includes a first layer of 3D convolutional kernels that use 3D filters. This layer extracts both spatially correlated features in the x-y plane along with any specific signatures in the frequency axis (z dimension), which can be used as input to succeeding layers.

[0280] The illustrated embodiment provides numerous advantages, including the ability to directly process compressed visual data in an efficient manner, thus eliminating the need to decompress the data before analyzing its contents (e.g., using a deep learning neural network). In this manner, the overall computational complexity of visual analytics can be reduced. Moreover, because compressed or DCT domain data is quantized and thus represented using a more compact form than the original visual data (e.g., video frame), the overall CNN complexity may be further reduced compared to a conventional pixel-domain CNN. For example, with respect to visual data (e.g., images or video) compressed in certain compression formats such as JPEG or M-JPEG, the DCT coefficients are quantized, and typically the highest frequency components may be zeroed out by the quantization. Thus, the total volume of non-zero data processed by the CNN is reduced compared to the original image data. Accordingly, based on the data volume reduction of the compressed data (e.g., due to DCT coefficient quantization), the CNN complexity may be further reduced, and the training speed of convergence may improve.

[0281] FIGS. **36** and **37** illustrate example embodiments of visual analytics pipelines **3600** and **3700** that perform visual analytics on compressed visual data (e.g., using the compression-based CNN embodiments described throughout this disclosure). As shown by these FIGURES, the decoding or decompression step in the visual analytics pipeline is optional and/or may be omitted entirely. For example, as shown by visual analytics pipeline **3600** of FIG. **36**, the original pixel domain data **3602** is first compressed by a video encoder **3610** (e.g., prior to transmission over a network), and the compressed data **3604** may optionally be partially decompressed by a video decoder **3620** before performing visual analytics **3630** on the fully or partially compressed data **3606**. Similarly, as shown by visual analytics pipeline **3700** of FIG. **37**, the original pixel domain data **3702** is first compressed by a video encoder **3710** (e.g., prior to transmission over a network), and visual analytics (e.g., image classification) **3720** is then directly performed on the compressed data **3704**.

[0282] FIG. **38** illustrates a performance graph **3800** showing the precision of a CNN trained using compressed visual data (e.g., 4×4 transform DCT inputs), such as the compression-based CNNs described throughout this disclosure.

[0283] FIG. **39** illustrates a flowchart **3900** for an example embodiment of context-aware image compression. In some embodiments, flowchart **3900** may be implemented using the embodiments and functionality described throughout this disclosure.

[0284] Today, many people rely on the cloud for storing or backing up their photos. Typically, photos are stored as individually compressed files or units. In the current computing era, however, that approach is often inefficient. For example, people increasingly use their mobile devices to take photos, and each new generation of mobile devices are updated with cameras that support more and more megapixels, which results in larger volumes of photos that require more storage space. Moreover, people often capture multiple photos of the same object or scene during a single occasion, which often results in a close temporal correlation among those photos, along with substantial redundancy. Accordingly, due to the redundancy across similar photos, individually compressing and storing each photo can be an inefficient approach. For example, traditionally, each photo is compressed and saved independently using a particular image compression format, such as JPEG. By compressing each photo individually, however, current approaches fail to leverage the inter-picture correlations between groups of similar photos, and thus more storage space is required to store the photos. For example, two photos that are nearly identical would still require double the storage of a single photo.

[0285] Accordingly, in the illustrated embodiment, groups of similar or related photos are compressed and stored more efficiently. For example, context information associated with photos is extracted and used to identify similar or related photos, and similar photos are then compressed jointly as a group. The contextual information, for example, could be used to identify a group of pictures from a single user that were taken very close in time and/or at the same location. As another example, the contextual information could be used to identify a group of pictures taken by different users but at the same location. Accordingly, the identified group of similar photos may be compressed using video coding in

order to leverage the inter-photo correlations and ultimately compress the photos more efficiently. In this manner, compressing related or correlated images using video compression rather than standard image compression can significantly reduce the storage space required for the photos (e.g., 2-5 times less storage space in some cases). Accordingly, this approach can be used to save or reduce storage in the cloud.

[0286] The flowchart may begin at block **3902** by first obtaining a new photo. In some cases, for example, the new photo could be captured by the camera of a mobile device. In other cases, however, any type of device or camera may be used to capture the photo.

[0287] The flowchart may then proceed to block **3904** to collect context information associated with the new photo. For example, when a photo is newly captured (e.g., by a mobile device), corresponding context information associated with the photo is collected, such as a timestamp, GPS coordinates, device orientation and motion states, and so forth.

[0288] The flowchart may then proceed to block **3906** to determine if a matching master photo can be identified for the new photo. In some embodiments, for example, the context information of the new photo is compared to the context information of other previously captured master photos to determine whether the new photo is closely correlated to any of the existing master photos. For example, if the photo is taken in the same location, within a certain amount of time, and with little phone movement compared to a master photo, it is likely that the new photo is highly correlated with the master photo. Further, in some embodiments, image feature matching techniques can then be applied to confirm the photo correlation. In some embodiments, for example, a scale-invariant feature transform (SIFT) may be used to determine whether a pair of photos are sufficiently correlated or matching.

[0289] If a matching master photo is identified at block **3906**, the flowchart may then proceed to block **3908** to encode the new photo with the matching master photo. In some embodiments, for example, a video codec (e.g., H.264) may be used to compress the new photo as an inter-frame associated with the master photo. For example, video codecs typically provide inter-frame encoding, which effectively utilizes the temporal correlation between similar images to improve the coding efficiency.

[0290] In some embodiments, a master photo may include any photo that is compressed without reference to other parent or related images, while a slave photo may include any photo that is compressed with reference to a master or parent image (e.g., using inter-frame mode of a video codec). Accordingly, a slave photo must efficiently record or correlate relevant information of its master photo, so that when the slave photo needs to be decoded for display of the entire image, the associated master photo can be quickly identified.

[0291] If a matching master photo is NOT identified at block **3906**, the flowchart may then proceed to block **3910** to encode the new photo by itself. For example, when the new photo does not match any of the existing master photos, the new photo is encoded without referencing any other photos, and the flowchart may then proceed to block **3912** to designate the new photo as a master photo, allowing it to potentially be compressed with other subsequently captured photos.

[0292] At this point, the flowchart may be complete. In some embodiments, however, the flowchart may restart and/or certain blocks may be repeated. For example, in some embodiments, the flowchart may restart at block **3902** to continue obtaining and compressing newly captured photos.

[0293] Privacy/Security

[0294] In distributed visual processing systems, it is important to implement effective privacy and security policies to protect sensitive visual data of underlying users or subjects (e.g., images or video with people's faces). Accordingly, in some embodiments, the visual fog architecture described throughout this disclosure may be implemented using a variety of privacy and security safeguards.

[0295] In some embodiments, for example, privacy-preserving distributed visual processing may be used in order to schedule or distribute vision workloads across available fog nodes in an efficient manner, while also adhering to any applicable privacy and/or security constraints.

[0296] Similarly, a multi-tiered storage approach may be used to store visual data in different locations and/or for different durations of time, depending on the particular level of sensitivity of the data. For example, the cloud may be used for long term storage of less sensitive or high-level visual data or metadata, while edge devices (e.g., on premise gateways) may be used for storage of highly sensitive visual data.

[0297] Moreover, certain vision operations may be implemented using privacy-preserving approaches. For example, for some vision applications (e.g., automated demographics identification), feature extraction and recognition may be implemented using cameras and sensors that capture top-down views rather than intrusive frontal views.

[0298] As another example, gateway cloud authentication may be used to securely authenticate gateways and/or other fog devices to the cloud using JSON web tokens.

[0299] As another example, wallets or distributed keys, along with MESH or GOSSIP based communication protocol, can be used to provide improved and more secure key management solutions.

[0300] Stream multiplexing may be used in application layer routing for streaming media, for example, by multiplexing visual sensors over multiple channels and introducing entropy to make channel prediction more difficult. For example, additional security can be provided by introducing entropy and other noise (e.g., chaff signals) designed to complicate channel prediction, thus thwarting efforts of malicious actors to pick up on video feeds.

[0301] As another example, a self-sovereign blockchain can be used to provide multi-tenant device identification. For example, the blockchain can be used to handle the orchestration and acceptance of device identities across multiple visual fog networks (e.g., even for legacy systems), thus allowing devices to assert their identity without relying on third party or centralized services. A self-sovereign blockchain can similarly be used for other purposes, such as managing a collection of distributed computing algorithms.

[0302] As another example, blockchain lifecycle management (e.g., managing the instantiation and lifecycle of blockchains) can be used to provide an additional level of security on blockchains used in a visual fog architecture. For example, blockchain lifecycle management can be used to ensure that a particular blockchain is implemented correctly and behaves as expected.

[0303] As another example, stakeholder management can be used to provide a set of protocols and frameworks to allow self-interests to be asserted, while arbitrating against conflicts in an equitable way.

[0304] FIGS. **40**A-C illustrate an example embodiment of a privacy-preserving demographic identification system **4000**. Identifying human demographic attributes (e.g., age, gender, race, and so forth) can be leveraged for a variety of use cases and applications. Example use cases include human-computer interaction, surveillance, business and consumer analytics, and so forth. In retail and healthcare segments, for example, defining a target audience and developing customer profiles has become a critical factor for successful brand strategy development.

[0305] In some embodiments, for example, computer vision and/or facial recognition technology may be used to identify human demographics. For example, demographics could be identified based on frontal and/or side facial features extracted using computer vision facial recognition technology. The use of frontal facial recognition technology in public, however, may implicate potential privacy concerns. Moreover, demographic identification is crucial across different domains and should not be limited to only frontal-based sensors and recognition techniques, particularly in the Internet-of-Things (IoT) era, which is projected to have over 20 billion connected devices by year 2020. Further, when limited to frontal-based vision sensors, it may be challenging to develop a demographics identification system that overcomes the person occlusion problem, while also providing wide processing viewing angles.

[0306] Accordingly, in the illustrated embodiment of FIGS. **40**A-C, privacy-preserving demographic identification system **4000** uses one or more top-view sensors **4015** to identify human demographics. In some embodiments, for example, either a single sensor **4015** or multiple sensors **4015** may be used to capture top-down views of humans, rather than conventional frontal views. Moreover, human demographics may then be identified based on features extracted from the top-down views captured by the sensors **4015**. In this manner, the use of top-view sensors **4015** enables human demographics to be automatically identified while preserving privacy, providing wider sensor viewing angles, and reducing susceptibility to occlusion.

[0307] FIG. **40**A illustrates a high-level implementation of demographic identification system **4000**. In the illustrated embodiment, edge devices **4010** include multiple sets of top-view sensors **4015***a-c* that are used for sensing humans. For example, each set of top-view sensors **4015***a-c* may include one or more sensors that are capable of capturing information about their surrounding environment. The information captured by top-view sensors **4015***a-c* is then processed in the fog **4020** to detect humans and identify their demographics. The contextual information extracted by the fog **4020** (e.g., human demographics) may then be transmitted to the cloud **4030** for further analytics, such as people profiling or generating heat maps.

[0308] FIG. **40**B illustrates an example of a set of top-view sensor(s) **4015** associated with demographic identification system **4000** of FIG. **40**A. As shown in the illustrated example, top-view sensors **4015** include a collection of one or more sensors positioned above an area that is accessible to humans **4002**. In some embodiments, for example, top-view sensors **4015** could be mounted to the ceiling of a retail store near the entrance. Moreover, top-view sensors **4015** can include any type and/or combination of sensor(s), such as a vision camera, infrared camera, light detection and ranging (LiDAR) sensor, and so forth. In this manner, top-view sensors **4015** can be used to capture top-view representations of humans **4002** that pass below the sensors. Moreover, as described further with respect to FIG. **40**C, the top-view representations captured by top-view sensors **4015** can then be processed further to identify the demographics of humans **4002** captured by the sensors.

[0309] FIG. **40**C illustrates an example of the demographics identification process performed by the fog **4020** in demographic identification system **4000** of FIG. **40**A. In the illustrated example, the demographics identification process involves (i) training a demographics classification model, and (ii) identifying demographic information using the trained demographics classification model with top-view sensor data as input.

[0310] The process of training the demographics classification model is illustrated by blocks **4021-4024**. At block **4021**, a training database of top-view human data must first be obtained or generated. In some embodiments, for example, the training database may include data captured by top-view sensors **4015**, such as camera images, infrared images, point clouds, and so forth. At block **4022**, features that are typically representative of human demographics are then selected/trained from the database using feature extraction methodologies, such as principal component analysis (PCA), discrete cosine transforms (DCT), machine learning (e.g., deep learning using a neural network), and so forth. At block **4023**, the selected/trained features are then provided as input to a process used to train a demographics classification model. At block **4024**, the trained demographics model is then saved in the fog **4020** for subsequent use during the demographics identification process, as described further below.

[0311] The process of identifying human demographics is illustrated by blocks **4025-4029**. At block **4025**, sensor data is captured by edge devices **4010** using one or more top-view sensor(s) **4015**, such as a vision camera, infrared camera, LiDAR sensor, and so forth. The raw sensor data (e.g., RGB images, thermal images, point clouds) is then transmitted from the edge **4010** to the fog **4020** in order to perform data pre-processing in the fog **4020** (e.g., on-premises), such as data transformations, de-noising, and so forth. At block **4026**, person detection is then performed on the pre-processed input stream. In some embodiments, for example, the pre-processed input stream is analyzed to determine if a person is captured in the underlying visual data. As an example, pre-processed image data from a top-view camera may be analyzed to determine if the image contains a person, and if so, the portion of the image that contains the person may be extracted. At block **4027**, features that are typically representative of human demographics are then selected or extracted from the detected person using feature extraction/machine learning techniques. At block **4028**, the extracted features from block **4027** and the pre-trained demographics model from block **4024** are then used by a demographics classifier to classify the demographic attributes of the detected person. At block **4029**, demographic information associated with the detected person is then identified based on the output of the demographics classifier.

[0312] The described embodiments of top-view demographics identification provide numerous advantages. As an

example, the described embodiments enable demographic information to be accurately identified based on top-down views of humans captured using a single- or multi-sensor approach. Compared to a frontal view approach, for example, a top-down or aerial perspective provides a wider angle of view for processing, reduces the problem of blocking or occlusion of people captured by the sensors, and preserves depth information associated with people and features captured and processed by the system. In addition, the described embodiments are less privacy-intrusive, as they only capture top views of people rather than other more intrusive views, such as frontal views. The described embodiments also identify demographic information based on permanent or lasting anthropometry features rather than features that may change or vary. Moreover, unlike motion-based detection approaches, the described embodiments are operable using only static views or images and do not require continuous image sequences or videos. Further, the described embodiments can be leveraged for a variety of use cases and applications, including retail, digital surveillance, smart buildings, and/or other any other applications involving human sensing, person identification, person re-identification (e.g., detecting/tracking/re-identifying people across multiple monitored areas), and so forth.

[0313] FIG. **53** illustrates a flowchart **5300** for an example embodiment of privacy-preserving demographics identification. In some embodiments, for example, flowchart **5300** may be implemented by demographics identification system **4000** of FIGS. **40**A-C.

[0314] The flowchart may begin at block **5302** by obtaining sensor data from a top-view sensing device. A top-view sensing device, for example, may be used to capture sensor data associated with the environment below the top-view sensing device (e.g., from a top-down perspective). In some embodiments, the top-view sensing device may include a plurality of sensors, including a camera, infrared sensor, heat sensor, laser-based sensor (e.g., LiDAR), and so forth.

[0315] The flowchart may then proceed to block **5304** to perform preprocessing on the sensor data, such as data transformations, filtering, noise reduction, and so forth. In some embodiments, for example, the raw sensor data may be transmitted to and/or obtained by a processor that is used to perform the preprocessing. For example, the preprocessing may be performed by an edge processing device at or near the network edge (e.g., near the top-view sensing device), such as an on-premise edge gateway.

[0316] The flowchart may then proceed to block **5306** to generate a visual representation of the environment below the top-view sensing device. The visual representation, for example, may be generated using the sensor data captured by the top-view sensing device (e.g., camera images, infrared images, point clouds, and so forth). In some embodiments, for example, the visual representation may be a three-dimensional (3D) representation or mapping of the environment from a top-down perspective. Moreover, in some embodiments, the visual representation may be generated at or near the network edge (e.g., near the top-view sensing device). For example, in some embodiments, an edge processing device (e.g., an on-premise edge gateway) may be used to generate the visual representation.

[0317] The flowchart may then proceed to block **5308** to determine whether a person is detected in visual representation. For example, if a person was located under the top-view sensing device when the sensor data was captured, then the visual representation generated using the sensor data may include a representation of the person from a top-view perspective. Accordingly, the visual representation may be analyzed (e.g., using image processing techniques) to determine whether it contains a person. In some embodiments, for example, the person detection may be performed at or near the network edge (e.g., near the top-view sensing device) by an edge processing device (e.g., an on-premise edge gateway).

[0318] If it is determined at block **5308** that a person is NOT detected in the visual representation, the flowchart may proceed back to block **5302** to continue obtaining and processing sensor data until a person is detected.

[0319] If it is determined at block **5308** that a person is detected in the visual representation, however, the top-view representation of the person may be extracted from the visual representation, and the flowchart may then proceed to block **5310** to identify one or more features associated with the person. In some embodiments, for example, the top-view representation of the person may be analyzed to identify or extract anthropometric features associated with the person (e.g., features or measurements associated with the size and proportions of the person). For example, in some embodiments, the anthropometric features may be identified by performing feature extraction using an image processing technique, such as a discrete cosine transform (DCT), principal component analysis (PCA), machine learning technique, and so forth. Moreover, in some embodiments, the feature identification or extraction may be performed at or near the network edge (e.g., near the top-view sensing device) by an edge processing device (e.g., an on-premise edge gateway).

[0320] The flowchart may then proceed to block **5312** to identify demographic information associated with the person (e.g., age, gender, race) based on the identified features. In some embodiments, for example, a machine learning model may be trained to recognize demographic information based on human anthropometric features. In this manner, the machine learning model can be used to classify the identified features of the person to recognize the associated demographic information.

[0321] In some embodiments, the demographics identification may be performed at or near the network edge (e.g., near the top-view sensing device) by an edge processing device (e.g., an on-premise edge gateway). Moreover, in some embodiments, the edge processing device may transmit the demographics information (e.g., using a communication interface) to a cloud processing device to perform further analytics, such as generating a heat map or a people profile.

[0322] At this point, the flowchart may be complete. In some embodiments, however, the flowchart may restart and/or certain blocks may be repeated. For example, in some embodiments, the flowchart may restart at block **5302** to continue obtaining and processing sensor data from a top-view sensing device.

[0323] FIGS. **41-43** illustrate an example embodiment of privacy-preserving distributed visual data processing.

[0324] In visual computing, multi-target multi-camera tracking (MTMCT) and target re-identification (ReID) are some of the most common workloads across different use cases. MTMCT involves tracking multiple objects across multiple views or cameras, while ReID involves re-identifying an object (e.g., by extracting robust features) even

after the object undergoes significant changes in appearance. For example, in retail, MTMCT is often used to track shoppers within a store, while ReID may be used to extract and summarize robust features of shoppers so they can later be re-identified (e.g., using MTMCT) in different circumstances, such as when a shopper has a significant change in appearance or visits a different store.

[0325] Currently, there are no coherent end-to-end (E2E) solutions for performing MTMCT and ReID that are scalable to large-scale visual computing systems (e.g., with tens of thousands of camera streams or more). In particular, bandwidth limitations render it challenging to deploy such a system in a conventional cloud computing paradigm where cameras send continuous video streams to the cloud for processing. For example, due to the large volume of video data generated by such systems, it is not feasible to funnel all of that data to the cloud for processing. On the other hand, it is unlikely that edge devices near the source of the video data are capable of processing a complete visual processing workload in real time.

[0326] Moreover, privacy is also a challenge in scaling out such a system, as sending visual data to the cloud for processing may implicate privacy concerns. For example, in order to preserve customer privacy, many retailers will not allow any video or images to be transmitted out of their stores.

[0327] Accordingly, FIGS. **41-43** illustrate an embodiment that solves the problem of scaling out visual computing systems with MTMCT and ReID capabilities in a privacy-preserving manner. The illustrated embodiment presents an edge-to-edge (E2E) architecture for performing MTMCT and ReID across edge devices, gateways, and the cloud. The architecture is scalable and privacy-preserving, and can be easily generalized to many vertical applications or use cases, such as shopper insights in retail, people searching in digital security and surveillance, player tracking and replays in sports, and so forth.

[0328] In some embodiments, for example, vision workloads may be scheduled and executed across visual fog nodes based on specified privacy constraints. As an example, privacy constraints for an MTMCT and/or ReID workload may require tasks that output pictures with faces to remain on-premises (e.g., neither the tasks nor their output are assigned or transmitted beyond the premise or to the cloud), be anonymized (e.g., face-blurred), and/or be deployed only on devices with enhanced link security.

[0329] In some embodiments, for example, rather than funneling every bit of visual data to the cloud for processing, intelligent decisions can be made regarding how visual data and workloads are processed and distributed across a visual computing system. Based on the privacy requirements of a particular visual application, for example, a privacy boundary can be defined within the end-to-end paradigm of a visual computing system in order to achieve performance efficiency while also preserving privacy.

[0330] In some embodiments, for example, job partitioning can be used to partition a visual analytics workload into a directed acrylic graph (DAG) with vertices that represent primitive visual operations and edges that represent their dependencies. In this manner, the graph can be used to represent the various tasks and associated dependencies for a particular workload. Moreover, a privacy policy can be defined separately for each dependency. Similarly, a device connectivity graph can be used to represent the various devices and their connectivity in the edge-to-cloud paradigm, and a privacy level agreement (PLA) can be established for each edge of connectivity in the graph. In this manner, the edge-to-cloud architecture can be implemented to include a coherent management interface that performs end-to-end workload distribution without compromising privacy. For example, using the job partitioning approach described above, workload distribution effectively becomes a mapping problem of assigning the tasks of a workload onto devices in the edge-to-cloud paradigm. In some embodiments, for example, a global scheduler can be used to determine an optimal mapping between tasks and devices in order to maximize performance while preserving privacy constraints.

[0331] FIG. **41** illustrates an example visual workload graph **4100** for performing MTMCT and ReID. Example workload **4100** includes a plurality of tasks, including pre-processing **4102**, detection **4104**, tracking **4106**, matching **4108**, and database access **4110**. Further, the dependencies between these various tasks are represented by the solid and dotted lines in the illustrated example. Moreover, the solid lines represent unrestricted access or transmission of the original visual data, while the dotted lines represent restricted or privacy-preserving access or transmission (e.g., transmitting only visual metadata, such as feature vectors). In this manner, a privacy policy can be defined for the workload, for example, by specifying whether each task has unrestricted access or restricted access to the original visual data.

[0332] FIG. **42** illustrates an example of an edge-to-cloud device connectivity graph **4200**. In the illustrated example, graph **4200** illustrates the connectivity between various devices of a 3-tier edge-to-cloud network, which includes cameras **4210***a-c*, gateways **4220***a-b*, and the cloud **4230**. In particular, the device connectivity is illustrated for both edge-to-cloud communications (e.g., camera to gateway to cloud) as well as peer-to-peer communications (e.g., gateway-to-gateway). Moreover, the connectivity between the respective devices is represented using solid and dotted lines. For example, the solid lines represent high-security connectivity links, while the dotted lines represent limited-security connectivity links. In this manner, a privacy policy or privacy level agreement (PLA) can be defined for an edge-to-cloud paradigm, for example, by specifying the requisite security for each edge of connectivity in the graph.

[0333] FIG. **43** illustrates a privacy-preserving workload deployment **4300**. In particular, workload deployment **4300** illustrates an example deployment of the workload **4100** of FIG. **41** on edge-to-cloud network **4200** of FIG. **42**.

[0334] In the illustrated example, privacy is treated as an explicit constraint when performing task-to-device mapping to deploy the workload. In some embodiments, for example, workloads can be represented in linear forms to enable the mapping problem to be solved efficiently using state of the art integer linear programming (ILP) solvers.

[0335] In some embodiments, for example, when scheduling a particular workload on an edge-to-cloud network, the workload and the edge-to-cloud network may each be represented using a graph, such as a directed acrylic graph (DAG). For example, the workload and its underlying tasks may be represented by a workload or task dependency graph $G_T=(V_T, E_T)$, where each vertex $v \in V_T$ represents a task, and each edge $(u, v) \in E_T$ represents a dependency between task u and task v. Similarly, the edge-to-cloud network may be

represented by a network or device connectivity graph $GD=(V_D, E_D)$, where each vertex $v \in V_D$ represents a device in the network, and each edge $(u, v) \in E_D$ represents the connectivity from device u to device v.

[0336] Moreover, the privacy policy (PP) for each task dependency in the workload graph may be defined using a PP function $p: E_T \rightarrow \mathbb{N}$, such that the smaller the number (N), the more vulnerable the data transmission. Similarly, the privacy level agreement (PLA) for each connectivity link in the device connectivity graph may be defined using a PLA function $s: E_D \rightarrow \mathbb{N}$, such that the smaller the number ($\mathbb{N}$), the more secure the link.

[0337] In this manner, based on the privacy policy (PP) and privacy level agreement (PLA) functions, a privacy constraint (PC) can be defined as $s(d) \leq p(e), \forall e \in E_T, d \in f(e)$, where $f: E_T \rightarrow x_{i=0}^{k} E_D$ is the mapping function from a particular workload to the edge-to-cloud paradigm. Essentially, f maps an edge in a workload graph to a path in an edge-to-cloud connectivity graph. For example, in the context of visual fog computing, f is a scheduling function that determines the particular fog devices that the tasks of a workload should be assigned to, along with the particular network connectivity links between pairs of fog devices that should be used for the data transmissions. Accordingly, the above privacy constraint (PC) requires the privacy level agreement (PLA) of a particular connectivity link to be capable of accommodating the privacy policy (PP) of a particular data transmission sent over that connectivity link. For example, in some embodiments, a data transmission of PP level 1 (unrestricted access) can only map to a link of PLA level 1 (high security), while a data transmission of PP level 2 (privacy-preserving) can map to connectivity links of PLA level 1 (high security) and PLA level 2 (limited security).

[0338] Moreover, in some embodiments, a visual fog schedule that adheres to the above privacy constraint (PC) can be determined using integer linear programming (ILP). Integer linear programming (ILP) is a mathematical optimization or feasibility technique for solving or optimizing a mathematical model represented by linear relationships. In particular, ILP can be used to optimize a linear objective function, subject to additional linear equality and linear inequality constraints. In some cases, for example, an ILP problem can be expressed as follows:

[0339] minimize: $c^T x$ (objective term)

[0340] subject to: $Ax \leq b$ (inequality constraint)

[0341] $Cx=d$ (equality constraint)

[0342] and: $x \in \{0, 1\}^K$ (binary constraint).

[0343] Moreover, this ILP model can be used to determine an optimal schedule f that satisfies a specified objective (e.g., total network utilization), while also adhering to other additional constraints, such as a privacy constraint and any other device, network, or mapping constraints. For example, when using the example ILP model above to perform visual fog scheduling, x presents the collection of possible schedules f, K is the length of x, the objective term presents a scheduling objective to be minimized (e.g., total network utilization), and the inequality/equality constraints present any additional constraints, such as device, network, mapping, and/or privacy constraints. The above privacy constraint (PC), for example, can be presented as an inequality constraint of the ILP problem.

[0344] FIG. **54** illustrates a flowchart **5400** for an example embodiment of privacy-preserving distributed visual processing. In some embodiments, for example, flowchart **5400** may be implemented using the visual computing embodiments described throughout this disclosure (e.g., the privacy-preserving distributed visual processing techniques of FIGS. **41-43** and/or the visual computing architecture described throughout this disclosure).

[0345] The flowchart may begin at block **5402** by identifying a new workload. In some embodiments, for example, the new workload may include a plurality of tasks associated with processing sensor data captured by one or more sensors. For example, in some embodiments, the sensor data may be visual data captured by one or more vision-based sensors (e.g., a camera, infrared sensor, and/or laser-based sensor).

[0346] The flowchart may then proceed to block **5404** to generate a workload graph based on the workload. In some embodiments, for example, the workload graph may include information associated with the underlying tasks of the workload, along with the task dependencies among those tasks.

[0347] The flowchart may then proceed to block **5406** to generate or identify a device connectivity graph. In some embodiments, for example, the device connectivity graph may include device connectivity information associated with a plurality of processing devices, such as edge, cloud, and/or intermediary network processing devices. The device connectivity information, for example, may include information associated with the device connectivity links among the plurality of processing devices.

[0348] The flowchart may then proceed to block **5408** to identify a privacy policy associated with the workload and/or its underlying tasks. In some embodiments, for example, the privacy policy may comprise privacy requirements associated with the task dependencies among the workload tasks.

[0349] The flowchart may then proceed to block **5410** to identify privacy level information associated with the plurality of processing devices. In some embodiments, for example, the privacy level information may include privacy levels provided by the device connectivity links among the plurality of processing devices. Moreover, in some embodiments, the privacy level information may be specified by a privacy level agreement.

[0350] The flowchart may then proceed to block **5412** to identify a privacy constraint for workload scheduling based on the privacy policy and the privacy level information. In some embodiments, for example, the privacy constraint may require the privacy level of a particular connectivity link to be capable of accommodating the privacy policy of any task dependency mapped to that connectivity link for data transmission.

[0351] The flowchart may then proceed to block **5414** to determine a workload schedule. The workload schedule, for example, may include a mapping of the workload onto the plurality of processing devices. Moreover, in some embodiments, the workload schedule may be determined based on the privacy constraint, the workload graph, and the device connectivity graph. For example, in some embodiments, the workload schedule may be determined by solving an integer linear programming model based on the privacy constraint, the workload graph, and the device connectivity graph (e.g., as described in connection with FIGS. **41-43**). In this manner, a resulting workload schedule is determined in a manner that adheres to the privacy constraint. Moreover, in

some embodiments, a machine learning model may be used to optimize privacy-constrained workload scheduling.

[0352] In some embodiments, the resulting workload schedule may then be distributed to the plurality of processing devices (e.g., via a communication interface) in order to execute the workload.

[0353] At this point, the flowchart may be complete. In some embodiments, however, the flowchart may restart and/or certain blocks may be repeated. For example, in some embodiments, the flowchart may restart at block **5402** to continue scheduling new workloads.

[0354] FIGS. **44-46** illustrate example embodiments of self-sovereign device identification for distributed computing networks. In some embodiments, for example, a fog node (e.g., IoT sensor, actuator, camera, controller, gateway, and/or any other type of fog node) may be a "multi-tenant" node that is capable of participating in multiple different distributed computing networks (e.g., visual fog networks). Moreover, certain networks may require a new fog node to be "on-boarded" or "commissioned" before the fog node is allowed to access each network (e.g., using the onboarding/commissioning protocols of the Open Connectivity Foundation (OCF) and/or Intel's Secure Device Onboard (SDO) technology). Many visual computing solutions, however, may assume that ownership of a node is singular, meaning each node has only one owner. Accordingly, ownership disputes may arise from a multi-tenant fog node's participation in multiple fog networks. The true or original owner of a multi-tenant fog node, however, has an interest in avoiding these ownership disputes. Accordingly, many visual computing solutions are unsuitable for multi-tenant fog nodes, which may participate in multiple fog networks while also abiding by each network's onboarding or commissioning protocols (e.g., as defined by OCF or Intel SDO).

[0355] Accordingly, in the illustrated embodiments, a multi-tenant fog node can use a self-sovereign device identity in order to allow the node owner to retain an assertion of ownership even when the fog node participates in, or roams to, other fog networks. In some embodiments, for example, a self-sovereign identity blockchain may be used to register the identities of fog nodes or devices. A blockchain, for example, may be a dynamic list of records or blocks that are linked and/or secured using cryptographic approaches. In some embodiments, for example, each block in a blockchain may include a hash pointer linking to a previous block, a timestamp, transaction data, and so forth. Accordingly, in some embodiments, a blockchain can be used as a distributed ledger for recording transactions in an efficient, verifiable, and/or permanent manner. In visual computing, for example, before adding a device identifier for a new fog node, a blockchain may optionally be used to verify that the identifier has not been previously asserted by another node. Further, the public key used to verify the device identity of the fog node may also be contributed to the blockchain, allowing the device to later prove it is the rightful owner of its identity.

[0356] FIG. **44** illustrates an example embodiment of a distributed computing architecture **4400** with multi-tenant device identification. In the illustrated embodiment, architecture **4400** includes fog networks A and B **4410***a-b*, self-sovereign identity blockchain **4420**, and new fog device **4430**, as described further below.

[0357] A new fog device **4430** that is seeking to be used in multiple fog networks **4410**, but is not exclusive to any particular fog network, may not have sufficient resources or capabilities to create and maintain virtual sandbox environments for each of the fog networks. Moreover, each fog network **4410** may have a large set of its own local fog devices that are exclusive to that network and do not roam into other fog networks. Accordingly, reusing device identifiers may not pose a significant problem of duplicative identifiers until a new device **4430** with a conflicting identity roams into a particular fog network.

[0358] There is often a cost associated with changing the identity of a device, however, as credentials, access tokens, and application logic may be linked to the device identity. Moreover, the respective owners of devices with conflicting identifies have a self-interest in resolving the conflict (e.g., to avoid ownership disputes), but without bearing the cost. For example, the conflicting devices may respectively view each other as "foreign," and thus each device may want the other "foreign" device to bear the cost of an identity change. Accordingly, to resolve the opposing self-interests of devices with conflicting identities, a blockchain **4420** may be used to provide a fair algorithm for giving preference to a device for its use of an identity. In some embodiments, for example, the device that first registered a particular identity with the blockchain **4420** is given preference in the event of a conflict.

[0359] FIG. **45** illustrates an example call flow **4500** for performing name registration of a self-sovereign device identity. In some embodiments, for example, registration of a self-sovereign device identity may be performed before onboarding a new fog device onto a visual fog network. For example, prior to being on-boarded onto a visual fog network, a fog device may register its choice of device identity with a blockchain.

[0360] Moreover, the blockchain may have a policy for preventing duplicative identity registrations, for example, by first checking for duplicates and only allowing registration if no duplicates exist. For example, duplicative identity detection may be performed by blockchain processing nodes as a requirement for vetting transaction blocks used for identity registration. In the illustrated call flow **4500**, for example, each node performs the following steps:

[0361] (1) receive transaction request from new device: TX_{n+1}={S1, "A71C3" }, where S1=Sign_K_{alice}("A71C3");

[0362] (2) compute hash H1=SHA256("A71C3");

[0363] (3) search hash tree of transaction attributes, where $B_{x\text{-}poss}$=Search(TxTree, H1);

[0364] (4) IF $B_{x\text{-}poss}$="H1" THEN return ERROR_DUP_FOUND;

[0365] (5) ELSE IF $B_{x\text{-}poss}$=" " THEN add TX_{n+1} to the current block where CurrentBlock=[TX_{n+1}, TX_n, TX_{n-1}, . . . , TX_{n-m}];

[0366] (6) compute new current block hash BH=SHA256([TX_{n+1}, TX_n, TX_{n-1}, . . . , TX_{n-m}]);

[0367] (7) write BH to the blockchain at $B_{curr\text{-}pos}$(current position); and

[0368] (8) insert the tuple (H1, BH, $B_{x\text{-}poss}$) into TxTree.

[0369] In some embodiments, however, a less restrictive policy may be used, such as a policy that does not check for duplicates during identity or name registration, and instead relies on dispute resolution to resolve duplicative identities. For example, at the time a device is on-boarded onto a new fog network, the blockchain can be consulted to determine if the identifier has previously been used, and if so, conflict resolution can be performed. The advantages of a less

restrictive policy include improved performance and the ability to support mass registration workloads, among other examples.

[0370] FIG. **46** illustrates an example call flow **4600** for conflict resolution of self-sovereign device identities. In some circumstances, for example, it may be unnecessary to verify that a new device identifier is globally unique at the time of registration, and instead, conflicting identities may be addressed when a new device is on-boarded onto a local fog network and an existing device already has the same identity. Accordingly, in some embodiments, conflicting device identities on a particular fog network may be resolved using conflict resolution call flow **4600**. In the illustrated call flow **4600**, for example, a blockchain is used to resolve conflicts based on identity registration priority (e.g., the first device that registered a duplicative identity with the blockchain receives preference). Accordingly, this approach does not require device identifiers to be globally unique, but in the event multiple devices on the same fog network have the same identity, it requires one of the devices to select a different identifier when interacting with that particular network. Moreover, the dispute over which device should pay the cost of changing its identity is resolved using the blockchain. By way of comparison, FIG. **47** illustrates an example of device onboarding or commissioning in a visual fog network without employing conflict resolution.

[0371] In this manner, based on the illustrated embodiments of FIGS. **44-46**, device identity assertion can be performed at any time during manufacturing of a device, such as a system-on-a-chip (SoC) or any other type of computing chip, circuit, or device. Moreover, rather than an assertion of device "ownership," device identity assertion involves an assertion of identity ownership, where the device is the owner of the identity. Accordingly, any appropriate entity within the supply chain of a particular device (e.g., an original design manufacturer (ODM), original equipment manufacturer (OEM), distributor, retailer, value-added reseller (VAR), installer, or end customer) may assert the identity of a device based on the sophistication and capability of the particular entity.

[0372] FIGS. **48** and **49** illustrate example embodiments of algorithm identification for distributed computing using a self-sovereign blockchain.

[0373] Distributed computing interoperability depends on agreement among participating nodes regarding the particular algorithms used to process information at each node. In some cases, for example, algorithm agreement among nodes may depend on a central authority that manages a registry or database of algorithm identifiers. In this manner, distributed nodes must rely on the registry for selection of the appropriate algorithms, otherwise interoperability is not achieved.

[0374] This dependence on central authorities can lead to service disruptions, however, such as when a registry goes offline, a registry is slow to publish new algorithm identifiers (e.g., thus slowing the pace at which new algorithms can be deployed), a central authority becomes the target of politicizations (e.g., registration requests are held in ransom for processing fees, political favors, and/or other forms of manipulation that are not tied to the economics of the distributed computing application), and so forth. For example, these approaches are often highly centralized and may involve international or governmental institutions, which may be prone to politicizations and/or government regulation (e.g., net neutrality). Moreover, since agreement on which algorithms to use is fundamental to distributed computing, a centralized approach for managing algorithm identifiers can create an artificial bottleneck or choking point, and entities seeking to impose regulation or control can effectively leverage the centralized design to restrict or prevent interoperability among distributed computing nodes.

[0375] Accordingly, in the illustrated embodiments of FIGS. **48** and **49**, a blockchain is used to register a collection of distributed computing algorithms (e.g., using self-sovereign algorithm identifiers). In some embodiments, for example, the blockchain may process an algorithm registration request as a blockchain transaction, where the registrant selects a unique algorithm identifier and specifies the algorithm function. In various embodiments, the algorithm function may be specified in human-readable form (e.g., as a natural language explanation or pseudocode), machine-readable form, and/or machine-executable form. Moreover, as a condition or prerequisite to accepting the algorithm registration, the particular algorithm may be subjected to various levels of "certification" by blockchain processing nodes. In this manner, an algorithm may be accepted with progressive levels of assurance without altering the registered algorithm identifier.

[0376] Accordingly, the described embodiments allow anyone that discovers a useful distributed computing algorithm to make that algorithm known and available to a large community. Blockchain networks, for example, are presumed to be large in number and open to large communities of users. In this manner, members of the community can build distributed computing systems without being hindered by bureaucratic roadblocks and oversight. As a result, the time between algorithm development and practical deployment can be minimized.

[0377] FIG. **48** illustrates an example embodiment of a distributed computing architecture **4800** with self-sovereign algorithm identification. In the illustrated embodiment, architecture **4800** includes fog networks A and B **4810***a-b*, along with a self-sovereign blockchain **4820** for registering and identifying distributed computing algorithms **4430**. In some embodiments, for example, architecture **4800** could be used to register and/or identify algorithms used for visual fog computing.

[0378] As an example, if a useful distributed computing algorithm **4430** is invented, discovered, and/or improved upon in a first fog network (e.g., fog network A **4810***a*), the first fog network may register the new algorithm in a self-sovereign blockchain **4420** used for algorithm identification. The blockchain processing nodes of the blockchain **4420** may then progressively vet the algorithm in order to provide progressively stronger assurances regarding its legitimacy (e.g., based on the computational properties and outcome of the algorithm). Moreover, a second fog network (e.g., fog network B **4810***b*) may subsequently be notified of the availability of the new algorithm, and may determine whether the new algorithm has been adequately vetted (e.g., by consulting the vetting status of the algorithm in the blockchain **4420**). If the second fog network is satisfied with the vetting of the new algorithm, the second fog network may agree to use the algorithm. For example, in some embodiments, after the algorithm has been adequately vetted, the first fog network and second fog network may agree to begin using the new algorithm.

[0379] In some embodiments, the algorithm registration and vetting process may involve: (1) registration of a self-sovereign algorithm identifier (SSAI); (2) peer-review of a human-readable description of the algorithm; (3) machine analysis of a machine-readable representation of the algorithm (e.g., analysis by a logic processor to identify safe behavioral properties); and (4) execution of a machine-executable implementation of the algorithm (e.g., execution in a sandbox environment used to analyze expected behavior). Moreover, once a certain threshold (e.g., a majority) of blockchain processing nodes or evaluators achieve similar vetting results, the algorithm identity and its vetting criteria/results are recorded in a block of the blockchain **4420**.

[0380] FIG. **49** illustrates an example call flow **4900** for registering a distributed computing algorithm using a self-sovereign blockchain. In some embodiments, for example, an algorithm may be registered using a self-sovereign blockchain to facilitate use of the algorithm across one or more distributed or fog computing environments. Moreover, in some embodiments, the blockchain may leverage various levels of vetting to ensure the algorithm behaves as expected, and verify that the algorithm identifier is not already in use.

[0381] In the illustrated call flow **4900**, for example, each blockchain processing node performs the following steps:

[0382] (1) receive transaction request from new device: TX_{n+1}={S1, "91E21" }, where S1=Sign_K_{alice}("91E21", "Human-readable-description", "Machine-readable-description", "Machine-executable-implementation");

[0383] (2) optional algorithm vetting (e.g., peer-review of a human-readable algorithm description, logical analysis of a machine-readable algorithm description/representation, sandbox execution of a machine-executable algorithm form);

[0384] (3) compute hash H1=SHA256("91E21");

[0385] (4) search hash tree of transaction attributes, where $B_{x\text{-}poss}$=Search(TxTree, H1);

[0386] (5) IF $B_{x\text{-}poss}$="H1" THEN return ERROR_DUP_FOUND;

[0387] (6) ELSE IF $B_{x\text{-}poss}$=" " THEN add TX_{n+1} to the current block, where CurrentBlock=[TX_{n+1}, TX_n, TX_{n-1}, . . . , TX_{n-m}];

[0388] (7) compute new current block hash BH=SHA256([TX_{n+1}, TX_n, TX_{n-1}, . . . , TX_{n-m}]);

[0389] (8) write BH to the blockchain at $B_{curr\text{-}pos}$(current position); and

[0390] (9) insert the tuple (H1, BH, $B_{x\text{-}poss}$) into TxTree.

[0391] Once the vetting process completes, the blockchain contains a vetted and registered instance of the algorithm and its associated identifier. In this manner, distributed computing nodes may then begin using the algorithm (e.g., based on the algorithm identifier and optionally its machine-readable and/or machine-executable forms).

[0392] Applications

[0393] The visual fog architecture and embodiments described throughout this disclosure can be used for a variety of large-scale visual computing applications and use cases, such as digital security and surveillance, business automation and analytics (e.g., retail and enterprise), transportation (e.g., traffic monitoring, navigation, parking, infrastructure planning, security or amber alerts), education, video broadcasting and playback, artificial intelligence, and so forth.

[0394] As an example, the described embodiments could be used to implement wearable cameras for first responders that are capable of automatically detecting events or emergency situations and performing certain responsive measures, such as notifying the appropriate personnel, triggering recording of the event by related or nearby cameras, and so forth.

[0395] As another example, the described embodiments could be used to implement a digital surveillance and security (DSS) system with people search or facial recognition capabilities across visual data streams from multiple different cameras, sensors, and/or locations.

[0396] As another example, the described embodiments could be used to implement a digital surveillance and security (DSS) system with license plate identification and fraud detection capabilities (e.g., identifying a car with a license plate that does not match the corresponding vehicle record, identifying multiple cars with same license plate, and so forth).

[0397] As another example, the described embodiments could be used to provide customer insights and analytics (e.g., for retail shoppers), such as an intra-store shopper trip summary (e.g., a list of products or departments interacted with by a shopper), an inter-store shopper trip summary (e.g., identifying repeat customers by differentiating between new and returning customers as they enter a store with a single or multiple locations), and so forth.

[0398] Similarly, the described embodiments could be used to provide visualization of customer or shopper insights and analytics (e.g., visualizing a graph representation of visual metadata for human consumption).

[0399] As another example, the described embodiments could be used to perform automated demographics identification in a privacy-preserving manner (e.g., using top-view cameras or sensors for demographic mapping of gender, age, race, and so forth).

[0400] As another example, the described embodiments could be used to perform heat mapping in retail stores or other brick-and-mortar environments to generate a representation of the crowd (e.g., using top-view sensors or cameras and/or multi-modal crowd emotion heat mapping). In some embodiments, for example, heat mapping could be leveraged for optimization of store layouts, among other examples.

[0401] As another example, the described embodiments could be used to implement multi-modal real-time customer reviews. For example, customer reviews and/or customer satisfaction information could be collected and analyzed in real-time using multi-sensory data, which can be translated into quantitative customer-to-customer reviews for any products or in-store activities of a particular store or brick-and-mortar environment.

[0402] Similarly, the described embodiments could be used to implement multi-modal retailer-shopper double review, which may focus on collection and analysis of both product reviews from customers and customer reviews from retailers.

[0403] As another example, the described embodiments could be used for automated customer satisfaction analysis. For example, visual data could be used to measure customer satisfaction at check-out based on non-verbal communication or body language. In this manner, customer satisfaction can be automatically inferred without requiring manual customer feedback (e.g., via a button or survey).

[0404] As another example, the described embodiments could be used to monitor the effectiveness of employee-customer interactions. For example, visual data could be used to measure and track the effectiveness of communication between customers and salespeople with respect to finding desired products or items. In some embodiments, for example, visual data could be used to track users within a store, identify customer-employee contact and interactions, and monitor the employee and/or customer responses.

[0405] As another example, the described embodiments could be used to provide dynamic ambience environments by identifying contextual information (e.g., relationships or actions) within a group of people. For example, visual data could be used to identify individuals and their associated contextual information to determine whether they are part of the same group (e.g., based on physical proximity and/or corresponding movement), and if so, to identify various parameters or characteristics of the group (e.g., a family shopping together in a store).

[0406] As another example, the described embodiments could be used to implement double auction real-time bidding (RTB). In some embodiments, for example, visual data could be used to implement multi-shopper, multi-bidder real-time bidding (RTB) for brick-and-mortar retailers.

[0407] As another example, the described embodiments could be used to monitor and detect changes to store layouts based on visual data and/or sensors.

[0408] As another example, the described embodiments could be used for robotic inventory tracking and logistics (e.g., using stationary and/or moving cameras to track inventory of retail stores, warehouses, offices, and so forth).

[0409] As another example, the described embodiments could be used for robotic equipment inspection (e.g., using computer vision technology to inspect the safety and/or health of equipment in a factory, plant, warehouse, store, office, and so forth).

[0410] As another example, the described embodiments could be used to provide automated tipping recommendations, for example, based on multi-sensory inputs and/or visual data reflective of factors that typically impact customer tipping behavior.

[0411] As another example, the described embodiments could be used for workplace automation, such as workplace quality control, employee monitoring, and so forth. In some embodiments, for example, visual data could be used to analyze employee emotions in order to improve productivity.

[0412] As another example, the described embodiments could be used for education and/or automated learning (e.g., using visual data to analyze student behavior in the classroom or at home in order to provide further assistance when appropriate).

[0413] As another example, the described embodiments could be used for video playback, such as user-centric video rendering, focused replays, and so forth. For example, user-centric video rendering could be used to perform focused rendering on 360-degree video by analyzing what the user is focusing on, and performing no or low-resolution processing on portions of the video that are outside the focus area of the user (e.g., for virtual-reality (VR) and/or augmented-reality (AR) applications). As another example, focused video replays could be used to automatically focus the rendering of a video replay on an area of interest, such as the portion of a sports replay where most players are located.

[0414] As another example, the described embodiments could be used to train artificial intelligence systems. In some embodiments, for example, visual data could be used to automatically generate ground truth information that can be used to train artificial intelligence or machine learning models, such as deep learning neural networks.

[0415] These examples are merely illustrative of the limitless universe of visual applications and use cases that can be implemented using the visual fog architecture described throughout this disclosure.

[0416] The flowcharts and block diagrams in the FIGURES illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various aspects of the present disclosure. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function (s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order or alternative orders, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

[0417] The foregoing disclosure outlines features of several embodiments so that those skilled in the art may better understand various aspects of the present disclosure. Those skilled in the art should appreciate that they may readily use the present disclosure as a basis for designing or modifying other processes and structures for carrying out the same purposes and/or achieving the same advantages of the embodiments introduced herein. Those skilled in the art should also realize that such equivalent constructions do not depart from the spirit and scope of the present disclosure, and that they may make various changes, substitutions, and alterations herein without departing from the spirit and scope of the present disclosure.

[0418] All or part of any hardware element disclosed herein may readily be provided in a system-on-a-chip (SoC), including a central processing unit (CPU) package. An SoC represents an integrated circuit (IC) that integrates components of a computer or other electronic system into a single chip. The SoC may contain digital, analog, mixed-signal, and radio frequency functions, all of which may be provided on a single chip substrate. Other embodiments may include a multi-chip-module (MCM), with a plurality of chips located within a single electronic package and configured to interact closely with each other through the electronic package. In various other embodiments, the computing functionalities disclosed herein may be implemented in one or more silicon cores in Application Specific Integrated Circuits (ASICs), Field Programmable Gate Arrays (FPGAs), and other semiconductor chips.

[0419] As used throughout this specification, the term "processor" or "microprocessor" should be understood to include not only a traditional microprocessor (such as Intel's® industry-leading x86 and x64 architectures), but also graphics processors, matrix processors, and any ASIC, FPGA, microcontroller, digital signal processor (DSP), programmable logic device, programmable logic array (PLA), microcode, instruction set, emulated or virtual machine processor, or any similar "Turing-complete" device, combination of devices, or logic elements (hardware or software) that permit the execution of instructions.

[0420] Note also that in certain embodiments, some of the components may be omitted or consolidated. In a general sense, the arrangements depicted in the figures should be understood as logical divisions, whereas a physical architecture may include various permutations, combinations, and/or hybrids of these elements. It is imperative to note that countless possible design configurations can be used to achieve the operational objectives outlined herein. Accordingly, the associated infrastructure has a myriad of substitute arrangements, design choices, device possibilities, hardware configurations, software implementations, and equipment options.

[0421] In a general sense, any suitably-configured processor can execute instructions associated with data or microcode to achieve the operations detailed herein. Any processor disclosed herein could transform an element or an article (for example, data) from one state or thing to another state or thing. In another example, some activities outlined herein may be implemented with fixed logic or programmable logic (for example, software and/or computer instructions executed by a processor) and the elements identified herein could be some type of a programmable processor, programmable digital logic (for example, a field programmable gate array (FPGA), an erasable programmable read only memory (EPROM), an electrically erasable programmable read only memory (EEPROM)), an ASIC that includes digital logic, software, code, electronic instructions, flash memory, optical disks, CD-ROMs, DVD ROMs, magnetic or optical cards, other types of machine-readable mediums suitable for storing electronic instructions, or any suitable combination thereof.

[0422] In operation, a storage may store information in any suitable type of tangible, non-transitory storage medium (for example, random access memory (RAM), read only memory (ROM), field programmable gate array (FPGA), erasable programmable read only memory (EPROM), electrically erasable programmable ROM (EEPROM), or microcode), software, hardware (for example, processor instructions or microcode), or in any other suitable component, device, element, or object where appropriate and based on particular needs. Furthermore, the information being tracked, sent, received, or stored in a processor could be provided in any database, register, table, cache, queue, control list, or storage structure, based on particular needs and implementations, all of which could be referenced in any suitable timeframe. Any of the memory or storage elements disclosed herein should be construed as being encompassed within the broad terms 'memory' and 'storage,' as appropriate. A non-transitory storage medium herein is expressly intended to include any non-transitory special-purpose or programmable hardware configured to provide the disclosed operations, or to cause a processor to perform the disclosed operations. A non-transitory storage medium also expressly includes a processor having stored thereon hardware-coded instructions, and optionally microcode instructions or sequences encoded in hardware, firmware, or software.

[0423] Computer program logic implementing all or part of the functionality described herein is embodied in various forms, including, but in no way limited to, hardware description language, a source code form, a computer executable form, machine instructions or microcode, programmable hardware, and various intermediate forms (for example, forms generated by an HDL processor, assembler, compiler, linker, or locator). In an example, source code includes a series of computer program instructions implemented in various programming languages, such as an object code, an assembly language, or a high-level language such as OpenCL, FORTRAN, C, C++, JAVA, or HTML for use with various operating systems or operating environments, or in hardware description languages such as Spice, Verilog, and VHDL. The source code may define and use various data structures and communication messages. The source code may be in a computer executable form (e.g., via an interpreter), or the source code may be converted (e.g., via a translator, assembler, or compiler) into a computer executable form, or converted to an intermediate form such as byte code. Where appropriate, any of the foregoing may be used to build or describe appropriate discrete or integrated circuits, whether sequential, combinatorial, state machines, or otherwise.

[0424] In one example, any number of electrical circuits of the FIGURES may be implemented on a board of an associated electronic device. The board can be a general circuit board that can hold various components of the internal electronic system of the electronic device and, further, provide connectors for other peripherals. More specifically, the board can provide the electrical connections by which the other components of the system can communicate electrically. Any suitable processor and memory can be suitably coupled to the board based on particular configuration needs, processing demands, and computing designs. Other components such as external storage, additional sensors, controllers for audio/video display, and peripheral devices may be attached to the board as plug-in cards, via cables, or integrated into the board itself. In another example, the electrical circuits of the FIGURES may be implemented as stand-alone modules (e.g., a device with associated components and circuitry configured to perform a specific application or function) or implemented as plug-in modules into application specific hardware of electronic devices.

[0425] Note that with the numerous examples provided herein, interaction may be described in terms of two, three, four, or more electrical components. However, this has been done for purposes of clarity and example only. It should be appreciated that the system can be consolidated or reconfigured in any suitable manner. Along similar design alternatives, any of the illustrated components, modules, and elements of the FIGURES may be combined in various possible configurations, all of which are within the broad scope of this specification. In certain cases, it may be easier to describe one or more of the functionalities of a given set of flows by only referencing a limited number of electrical elements. It should be appreciated that the electrical circuits of the FIGURES and its teachings are readily scalable and can accommodate a large number of components, as well as

more complicated/sophisticated arrangements and configurations. Accordingly, the examples provided should not limit the scope or inhibit the broad teachings of the electrical circuits as potentially applied to a myriad of other architectures.

[0426] Numerous other changes, substitutions, variations, alterations, and modifications may be ascertained to one skilled in the art and it is intended that the present disclosure encompass all such changes, substitutions, variations, alterations, and modifications as falling within the scope of the appended claims.

Example Implementations

[0427] The following examples pertain to embodiments described throughout this disclosure.

[0428] One or more embodiments may include an apparatus, comprising: a processor to: identify a workload comprising a plurality of tasks; generate a workload graph based on the workload, wherein the workload graph comprises information associated with the plurality of tasks; identify a device connectivity graph, wherein the device connectivity graph comprises device connectivity information associated with a plurality of processing devices; identify a privacy policy associated with the workload; identify privacy level information associated with the plurality of processing devices; identify a privacy constraint based on the privacy policy and the privacy level information; and determine a workload schedule, wherein the workload schedule comprises a mapping of the workload onto the plurality of processing devices, and wherein the workload schedule is determined based on the privacy constraint, the workload graph, and the device connectivity graph; and a communication interface to send the workload schedule to the plurality of processing devices.

[0429] In one example embodiment of an apparatus, the processor to determine the workload schedule is further to solve an integer linear programming model based on the privacy constraint.

[0430] In one example embodiment of an apparatus, the plurality of tasks is associated with processing sensor data from one or more sensors.

[0431] In one example embodiment of an apparatus, the one or more sensors comprise one or more of: a camera; an infrared sensor; or a laser-based sensor.

[0432] In one example embodiment of an apparatus, the sensor data comprises visual data.

[0433] In one example embodiment of an apparatus, the workload graph further comprises information associated with a plurality of task dependencies among the plurality of tasks.

[0434] In one example embodiment of an apparatus, the privacy policy comprises a plurality of privacy requirements associated with the plurality of task dependencies.

[0435] In one example embodiment of an apparatus, the device connectivity information comprises information associated with a plurality of device connectivity links among the plurality of processing devices.

[0436] In one example embodiment of an apparatus, the privacy level information comprises a plurality of privacy levels associated with the plurality of device connectivity links.

[0437] One or more embodiments may include a system, comprising: a plurality of sensors to capture sensor data associated with an environment; a plurality of processing devices, wherein the plurality of processing devices comprises a plurality of edge processing devices and a plurality of cloud processing devices, and wherein the plurality of processing devices is to: identify a workload, wherein the workload comprises a plurality of tasks associated with processing the sensor data captured by the plurality of sensors; generate a workload graph based on the workload, wherein the workload graph comprises information associated with the plurality of tasks; identify a device connectivity graph, wherein the device connectivity graph comprises device connectivity information associated with the plurality of processing devices; identify a privacy policy associated with the workload; identify privacy level information associated with the plurality of processing devices; identify a privacy constraint based on the privacy policy and the privacy level information; determine a workload schedule, wherein the workload schedule comprises a mapping of the workload onto the plurality of processing devices, and wherein the workload schedule is determined based on the privacy constraint, the workload graph, and the device connectivity graph; and distribute the workload schedule to the plurality of processing devices.

[0438] In one example embodiment of a system, the plurality of processing devices to determine the workload schedule is further to solve an integer linear programming model based on the privacy constraint.

[0439] In one example embodiment of a system, the plurality of sensors comprises one or more of: a camera; an infrared sensor; or a laser-based sensor.

[0440] In one example embodiment of a system, the workload graph further comprises information associated with a plurality of task dependencies among the plurality of tasks.

[0441] In one example embodiment of a system, the privacy policy comprises a plurality of privacy requirements associated with the plurality of task dependencies.

[0442] In one example embodiment of a system, the device connectivity information comprises information associated with a plurality of device connectivity links among the plurality of processing devices.

[0443] In one example embodiment of a system, the privacy level information comprises a plurality of privacy levels associated with the plurality of device connectivity links.

[0444] One or more embodiments may include at least one machine accessible storage medium having instructions stored thereon, wherein the instructions, when executed on a machine, cause the machine to: identify a workload comprising a plurality of tasks; generate a workload graph based on the workload, wherein the workload graph comprises information associated with the plurality of tasks; identify a device connectivity graph, wherein the device connectivity graph comprises device connectivity information associated with a plurality of processing devices; identify a privacy policy associated with the workload; identify privacy level information associated with the plurality of processing devices; identify a privacy constraint based on the privacy policy and the privacy level information; determine a workload schedule, wherein the workload schedule comprises a mapping of the workload onto the plurality of processing devices, and wherein the workload schedule is determined based on the privacy constraint, the workload graph, and the device connectivity graph; and distribute the workload schedule to the plurality of processing devices.

[0445] In one example embodiment of a storage medium, the instructions that cause the machine to determine the workload schedule further cause the machine to solve an integer linear programming model based on the privacy constraint.

[0446] In one example embodiment of a storage medium, the plurality of tasks is associated with processing sensor data from one or more sensors.

[0447] In one example embodiment of a storage medium: the workload graph further comprises information associated with a plurality of task dependencies among the plurality of tasks; and the privacy policy comprises a plurality of privacy requirements associated with the plurality of task dependencies.

[0448] In one example embodiment of a storage medium: the device connectivity information comprises information associated with a plurality of device connectivity links among the plurality of processing devices; and the privacy level information comprises a plurality of privacy levels associated with the plurality of device connectivity links.

[0449] One or more embodiments may include a method, comprising: identifying a workload, wherein the workload comprises a plurality of tasks associated with processing sensor data from one or more sensors; generating a workload graph based on the workload, wherein the workload graph comprises information associated with the plurality of tasks; identifying a device connectivity graph, wherein the device connectivity graph comprises device connectivity information associated with a plurality of processing devices; identifying a privacy policy associated with the workload; identifying privacy level information associated with the plurality of processing devices; identifying a privacy constraint based on the privacy policy and the privacy level information; determining a workload schedule, wherein the workload schedule comprises a mapping of the workload onto the plurality of processing devices, and wherein the workload schedule is determined based on the privacy constraint, the workload graph, and the device connectivity graph; and distributing the workload schedule to the plurality of processing devices.

[0450] In one example embodiment of a method, determining the workload schedule comprises solving an integer linear programming model based on the privacy constraint.

[0451] In one example embodiment of a method: the workload graph further comprises information associated with a plurality of task dependencies among the plurality of tasks; and the privacy policy comprises a plurality of privacy requirements associated with the plurality of task dependencies.

[0452] In one example embodiment of a method: the device connectivity information comprises information associated with a plurality of device connectivity links among the plurality of processing devices; and the privacy level information comprises a plurality of privacy levels associated with the plurality of device connectivity links.

1.-**25**. (canceled)

26. A computing device to perform privacy-preserving workload scheduling across a computing infrastructure, comprising:

network interface circuitry to communicate over a network; and

processing circuitry to:

receive, via the network interface circuitry, a request to schedule a workload for execution across the computing infrastructure;

access a privacy policy associated with the workload, wherein the privacy policy indicates a plurality of privacy requirements for execution of the workload;

access a privacy level agreement associated with the computing infrastructure, wherein the privacy level agreement indicates a plurality of privacy levels provided across the computing infrastructure;

determine, based at least in part on the privacy policy and the privacy level agreement, a workload schedule for executing the workload, wherein the workload schedule assigns execution of the workload across a portion of the computing infrastructure; and

send, via the network interface circuitry, the workload schedule to the portion of the computing infrastructure assigned to execute the workload.

27. The computing device of claim **3**, wherein:

the workload comprises a plurality of tasks and a plurality of task dependencies among the plurality of tasks; and

the computing infrastructure comprises a plurality of processing devices and a plurality of device connectivity links among the plurality of processing devices.

28. The computing device of claim **27**, wherein:

the plurality of privacy requirements are required across the plurality of task dependencies of the workload; and

the plurality of privacy levels are provided across the plurality of device connectivity links of the computing infrastructure.

29. The computing device of claim **28**, wherein the workload schedule assigns execution of the plurality of tasks of the workload across a subset of the plurality of processing devices of the computing infrastructure.

30. The computing device of claim **29**, wherein the workload schedule maps the plurality of task dependencies of the workload across a subset of the plurality of device connectivity links of the computing infrastructure.

31. The computing device of claim **4**, wherein the processing circuitry to send, via the network interface circuitry, the workload schedule to the portion of the computing infrastructure assigned to execute the workload is further to:

send, via the network interface circuitry, the workload schedule to the subset of the plurality of processing devices of the computing infrastructure assigned to execute the plurality of tasks of the workload.

32. The computing device of claim **27**, wherein at least some of the plurality of tasks of the workload are to process sensor data captured by one or more sensors.

33. The computing device of claim **32**, wherein:

the one or more sensors comprise one or more cameras; and

the sensor data comprises visual data captured by the one or more cameras.

34. The computing device of claim **33**, wherein at least some of the plurality of privacy requirements are associated with processing the visual data captured by the one or more cameras.

35. The computing device of claim **26**, wherein the processing circuitry to determine, based at least in part on the privacy policy and the privacy level agreement, the workload schedule for executing the workload is further to:

solve an integer linear programming model based on the privacy policy associated with the workload and the privacy level agreement associated with the computing infrastructure; and

map the workload across the computing infrastructure based on a solution to the integer linear programming model.

36. At least one non-transitory machine-readable storage medium having instructions stored thereon, wherein the instructions, when executed on processing circuitry, cause the processing circuitry to:

receive, via network interface circuitry, a request to schedule a workload for execution across a computing infrastructure;

access a privacy policy associated with the workload, wherein the privacy policy indicates a plurality of privacy requirements for execution of the workload;

access a privacy level agreement associated with the computing infrastructure, wherein the privacy level agreement indicates a plurality of privacy levels provided across the computing infrastructure;

determine, based at least in part on the privacy policy and the privacy level agreement, a workload schedule for executing the workload, wherein the workload schedule assigns execution of the workload across a portion of the computing infrastructure; and

send, via the network interface circuitry, the workload schedule to the portion of the computing infrastructure assigned to execute the workload.

37. The storage medium of claim **36**, wherein:

the workload comprises a plurality of tasks and a plurality of task dependencies among the plurality of tasks; and

the computing infrastructure comprises a plurality of processing devices and a plurality of device connectivity links among the plurality of processing devices.

38. The storage medium of claim **7**, wherein:

the plurality of privacy requirements are required across the plurality of task dependencies of the workload; and

the plurality of privacy levels are provided across the plurality of device connectivity links of the computing infrastructure.

39. The storage medium of claim **38**, wherein the workload schedule assigns execution of the plurality of tasks of the workload across a subset of the plurality of processing devices of the computing infrastructure.

40. The storage medium of claim **39**, wherein the workload schedule maps the plurality of task dependencies of the workload across a subset of the plurality of device connectivity links of the computing infrastructure.

41. The storage medium of claim **39**, wherein the instructions that cause the processing circuitry to send, via the network interface circuitry, the workload schedule to the portion of the computing infrastructure assigned to execute the workload further cause the processing circuitry to:

send, via the network interface circuitry, the workload schedule to the subset of the plurality of processing devices of the computing infrastructure assigned to execute the plurality of tasks of the workload.

42. The storage medium of claim **8**, wherein:

at least some of the plurality of tasks of the workload are to process visual data captured by one or more cameras; and

at least some of the plurality of privacy requirements are associated with processing the visual data captured by the one or more cameras.

43. The storage medium of claim **36**, wherein the instructions that cause the processing circuitry to determine, based at least in part on the privacy policy and the privacy level agreement, the workload schedule for executing the workload further cause the processing circuitry to:

solve an integer linear programming model based on the privacy policy associated with the workload and the privacy level agreement associated with the computing infrastructure; and

map the workload across the computing infrastructure based on a solution to the integer linear programming model.

44. A method of performing privacy-preserving workload scheduling across a computing infrastructure, comprising:

receiving, via network interface circuitry, a request to schedule a workload for execution across the computing infrastructure;

accessing a privacy policy associated with the workload, wherein the privacy policy indicates a plurality of privacy requirements for execution of the workload;

accessing a privacy level agreement associated with the computing infrastructure, wherein the privacy level agreement indicates a plurality of privacy levels provided across the computing infrastructure;

determining, based at least in part on the privacy policy and the privacy level agreement, a workload schedule for executing the workload, wherein the workload schedule assigns execution of the workload across a portion of the computing infrastructure; and

sending, via the network interface circuitry, the workload schedule to the portion of the computing infrastructure assigned to execute the workload.

45. The method of claim **44**, wherein:

the workload comprises a plurality of tasks and a plurality of task dependencies among the plurality of tasks; and

the computing infrastructure comprises a plurality of processing devices and a plurality of device connectivity links among the plurality of processing devices.

46. The method of claim **45**, wherein:

the plurality of privacy requirements are required across the plurality of task dependencies of the workload; and

the plurality of privacy levels are provided across the plurality of device connectivity links of the computing infrastructure.

47. The method of claim **46**, wherein:

the workload schedule assigns execution of the plurality of tasks of the workload across a subset of the plurality of processing devices of the computing infrastructure; and

the workload schedule maps the plurality of task dependencies of the workload across a subset of the plurality of device connectivity links of the computing infrastructure.

48. The method of claim **45**, wherein:

at least some of the plurality of tasks of the workload are to process visual data captured by one or more cameras; and

at least some of the plurality of privacy requirements are associated with processing the visual data captured by the one or more cameras.

49. The method of claim **11**, wherein determining, based at least in part on the privacy policy and the privacy level agreement, the workload schedule for executing the workload comprises:

solving an integer linear programming model based on the privacy policy associated with the workload and the privacy level agreement associated with the computing infrastructure; and

mapping the workload across the computing infrastructure based on a solution to the integer linear programming model.

50. A system for performing privacy-preserving workload scheduling across a computing infrastructure, comprising:

means for receiving a request to schedule a workload for execution across the computing infrastructure;

means for accessing a privacy policy associated with the workload, wherein the privacy policy indicates a plurality of privacy requirements for execution of the workload;

means for accessing a privacy level agreement associated with the computing infrastructure, wherein the privacy level agreement indicates a plurality of privacy levels provided across the computing infrastructure;

means for determining, based at least in part on the privacy policy and the privacy level agreement, a workload schedule for executing the workload, wherein the workload schedule assigns execution of the workload across a portion of the computing infrastructure; and

means for sending the workload schedule to the portion of the computing infrastructure assigned to execute the workload.

* * * * *