FlowNL: Asking the Flow Data in Natural Languages

Jieying Huang, Yang Xi, Junnan Hu, and Jun Tao, *Member, IEEE*

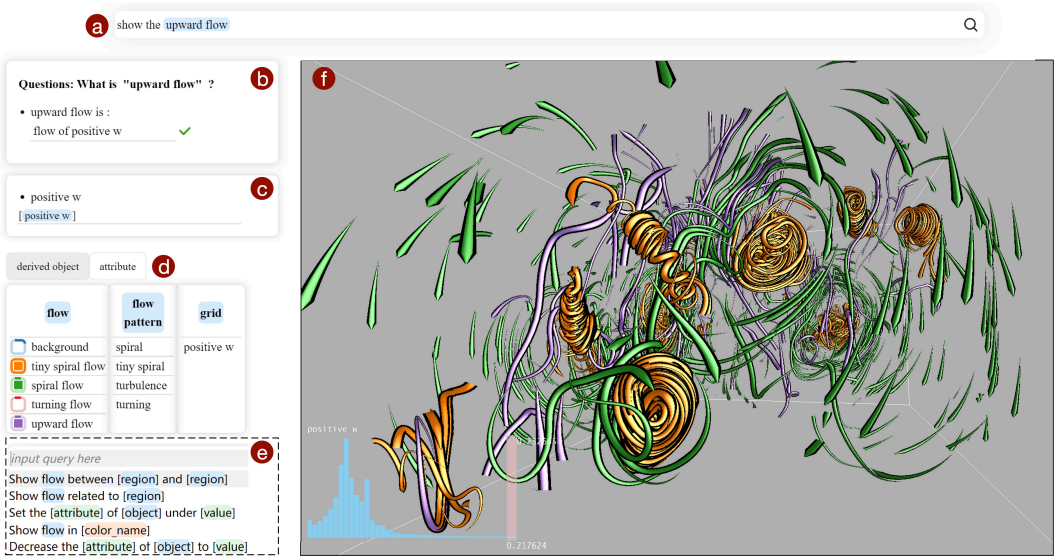


Fig. 1. The interface of FLowNL. (a) shows the query input box. (b) shows the dialog box to resolve unknown terms. (c) shows the query formula of a derived object. (d) shows primitive objects and their respective derived objects. The displayed object are indicated by small squares. (e) shows the suggested queries below the input box. (f) shows the streamlet visualization corresponding to three objects "tiny spiral flow", "spiral flow", and "upward flow".

Abstract— Flow visualization is essentially a tool to answer domain experts' questions about flow fields using rendered images. Static flow visualization approaches require domain experts to raise their questions to visualization experts, who develop specific techniques to extract and visualize the flow structures of interest. Interactive visualization approaches allow domain experts to ask the system directly through the visual analytic interface, which provides flexibility to support various tasks. However, in practice, the visual analytic interface may require extra learning effort, which often discourages domain experts and limits its usage in real-world scenarios. In this paper, we propose FlowNL, a novel interactive system with a natural language interface. FlowNL allows users to manipulate the flow visualization system using plain English, which greatly reduces the learning effort. We develop a natural language parser to interpret user intention and translate textual input into a declarative language. We design the declarative language as an intermediate layer between the natural language and the programming language specifically for flow visualization. The declarative language provides selection and composition rules to derive relatively complicated flow structures from primitive objects that encode various kinds of information about scalar fields, flow patterns, regions of interest, connectivities, etc. We demonstrate the effectiveness of FlowNL using multiple usage scenarios and an empirical evaluation.

Index Terms—Flow visualization, natural language interface, interactive exploration, declarative grammar.

1 Introduction

Flow visualization has been a central topic in scientific visualization for decades. The key to a successful flow visualization is to convey the information regarding flow structures of interest in the desired way. However, the definitions of "structures of interest" and the "desired way" often vary across domains, applications, or even experts' preference. The early approaches often place or select streamlines using fixed application-agnostic criteria, such as evenly-spacing between streamlines [18] and maximal information conveyed [56]. Other approaches target fixed types of features, such as saddles [49], critical points [35, 58], vortices [38]. These approaches are usually developed based on visualization experts' understanding of data or based on the communication between domain experts and visualization experts to determine what is desired. However, the approaches using fixed criteria may not meet the need of specific domains and applications, while the approaches targeting fixed types of features may not extend to other types of features or require significant development effort to do so.

The exploratory techniques emerge to customize the visualization for different features or applications. These approaches allow users to specify the streamlines related to the features of interest through graph-based interface [14,37,46], pattern query [51], predicates [20,39], and tangible interface [17]. Among these approaches, the graphical interface is most commonly used. However, although flexible, powerful graphical interfaces often require steep learning curves and significant comprehension effort. The tangible interface provides physical

Jieying Huang, Yang Xi, and Junnan Hu are with the School of Computer Science and Engineering, Sun Yat-sen University. E-mail: {huangjy85,yangxi3,hujn3}@mail2.sysu.edu.cn.
Jun Tao is with the School of Computer Science and Engineering, Sun Yat-sen University, National Supercomputer Center in Guangzhou, and Southern Marine Science and Engineering Guangdong Laboratory (Zhuhai). Email: taoj23@mail.sysu.edu.cn. He is the corresponding author.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

feedback for experts to interact with their data in a more intuitive manner. But due to the limited degree-of-freedom in the tangible interface interaction, it could be difficult to support sophisticated queries.

Inspired by recent advancements in natural language interface (NLI) for visualizing and exploring tabular data [27, 31, 44, 59], we propose FlowNL, a natural language interface for flow visualization, aiming to support various types of analysis tasks and reduce the learning and usage effort at the same time. FlowNL features a flexible scheme to *query flow structures* and *explore the structures and their connections* through a *dialogue mechanism*. Specifically, it supports the derivation of flow structures from the basic types of objects, named primitives. The primitive objects include the vector field, sampled streamlines, associated scalar fields, and other unstructured data points (e.g., critical points and geographic regions). FlowNL provides a series of simple operations to filter and combine the simple objects to derive more complicated ones. For example, the atmospherical phenomenon "typhoon" can be derived by combining "spirals" from filtering streamlines, "strong wind" from filtering the scalar field of velocity magnitude, and "in the west Pacific Ocean" from filtering the geographic locations. To derive new objects in natural interactions, FlowNL provides a dialogue mechanism. This allows users to define flow structures during a conversation, instead of defining all structures before a query. The defined flow structures are then visualized by an efficient streamlet visualization engine in an animated manner. Additionally, FlowNL supports the analysis of connections among the flow structures using a set of neighborhood operations. These operations extend the existing objects along the flow, which provides a Lagrangian view of the flow field.

Architecture. FlowNL is realized through three major components: *a natural language parser*, *a declarative language for flow visualization*, and *a visualization engine*. The natural language parser takes the natural language queries as input and translates the queries into specifications in the declarative language. The declarative language serves as an intermediate layer between the natural language and the flow visualization engine. It specifies how the objects should be combined to derive a new one, and how the objects should be visualized. The visualization engine performs the actual computation to derive objects and use them to guide the placement and removal of particles.

Contribution. Our contribution can be summarized as follows:

- We propose FlowNL, a natural language interface that translates the natural language queries to flow visualization results. Using the natural languages as queries is intuitive and greatly reduces the learning and usage effort.
- We design a declarative language that flexibly filters and combine the primitive objects (e.g., fields, streamlines, and features) to define structures of interest. The declarative language also specifies the connections among structure allowing the Lagrangian behaviors to be observed.
- We design an interface integrating the dialog box for natural language queries and the flow visualization. It also allows users to view and adjust the visualization styles of objects.

2 Related Work

Interactive flow visualization. Interactive techniques assist the visual exploration of flow data and customization of visualization results, providing the flexibility to handle different tasks. Several techniques are built on the graph representation to select streamlines or other flow structures, such as the flow web [57], streamline embedding [37], FlowGraph [28], semantic flow graph [46], and FlowNet [14]. While most of these approaches [14, 28, 37, 57] use distance metric to guide the generation of graph layout, semantic flow graph [46] uses the semantic information to group the elements of similar attributes for exploration. Streamline predicate [20, 39], IGScript [26], and tangible interface [17] also adopt a similar idea but assign the semantic information in different ways. Another commonly adopted strategy for interactive flow exploration is based on pattern matching or feature detection. These approaches often rely on the similarity measure of streamlines [47,51] and pattern matching for flow field regions [6] to identify similar patterns. More involved techniques are developed for specific applications and features, such as atmospheric front [19], PV banner [2], vortex [12, 13], and splat [33, 34].

Natural language interface. The natural language interface appeared in the information visualization field for around a decade. Most of the NLI approaches follow the same scheme: parsing the natural language queries, translating them into an intermediate form, such as explicit commands [45], SQL queries [9], VisFlow functions [59], and declarative specifications [32], and using the intermediate commands for visualization. Luo et al. [27] followed a similar scheme, but developed a transformer-based model for the translation. Several NLI systems feature interactive dialog between users and the systems, such as Articulate 2 [21], Eviza [42], and Evizeon [16]. DataTone [10] targets ambiguity in natural language. It resolves the ambiguity using algorithmic disambiguation coupled with interactive ambiguity widgets. Orko [44] supports multiple modalities of interaction including NLI. It uses a combination of grammar-based and lexicon-based parsing techniques to interpret the queries. The above work all support natural language query data for visualization. Cui et al. [8] proposed Text-to-Viz, which generated visualization results from multiple sets of collected visual elements instead of query results. Researchers also evaluated the NLI approaches to produce guidelines for future developments. Srinivasan et al. [43] designed several tasks to examine and compare five NLI systems, aiming to contrast them to reveal the challenges in designing NLIs. Tory et al. [50] designed an empirical study for their system, suggested approaches to interpret and respond to users' intent, and reveal how varying levels of system understanding might affect the user experience.

Declarative language for visualization. The use of declarative language in information visualization toolkits became popular in the last decade, such as D3 [3], Reactive Vega [41], Vega-Lite [40], ggplot2 [52], and GoTree [22]. Recently, the declarative language is also used to provide computational simplicity and build visual analytic systems. For example, Li and Ma [23] proposed P4, which generated WebGL programs in runtime to enable high-performance data processing and visualization. The authors later proposed P5 [24] that extended the data transformation and visualization capabilities for progressive analysis and visualization, and P6 [25] that used declarative language to combine interactive visualizations and machine learning.

The declarative languages also received attention from the scientific visualization community. Shih et al. [53] presented a declarative grammar for customizing volume visualization pipelines. Their grammar focuses on the needs of specification of DVR-based volume visualization. Wu et al. [54] designed DIVA, a declarative language for in situ data analysis and visualization, which made adaptive workflow development a simpler process. Liu et al. [26] proposed IGScript, a declarative language for interactive scientific data presentations. This is different from the previous works which often target the computation or rendering stages. Similar approaches use the boolean formula to specify transfer function for features in volume [5] and domain-specific language for volume processing [7, 36].

Comparison with the existing works. In terms of how the flow structure is represented, the semantic-based interactive approaches, such as semantic flow graph [46], streamline predicate [39], and IGScript [26] is most similar to our FlowNL. Our declarative specification can be considered as creating semantic labels by filtering and composing existing labels (predicates). However, our approach also extends these approaches in the granularity of specification. These approaches select flow structures at the streamline level. Therefore, they have limited power in finding structures that are only related to streamline segments (e.g., flow in high pollution regions). Additionally, FlowNL integrates the filtering and linking across multiple spaces, while the other approaches may need additional steps for this purpose. Finally, FlowNL provides a convenient interaction scheme using natural language, which is not available in previous approaches.

In terms of natural language interface, FlowNL shares similar framework with previous techniques: translating natural language into an intermediate language that customizes the visualization. But unlike the existing techniques, FlowNL targets a significantly different domain

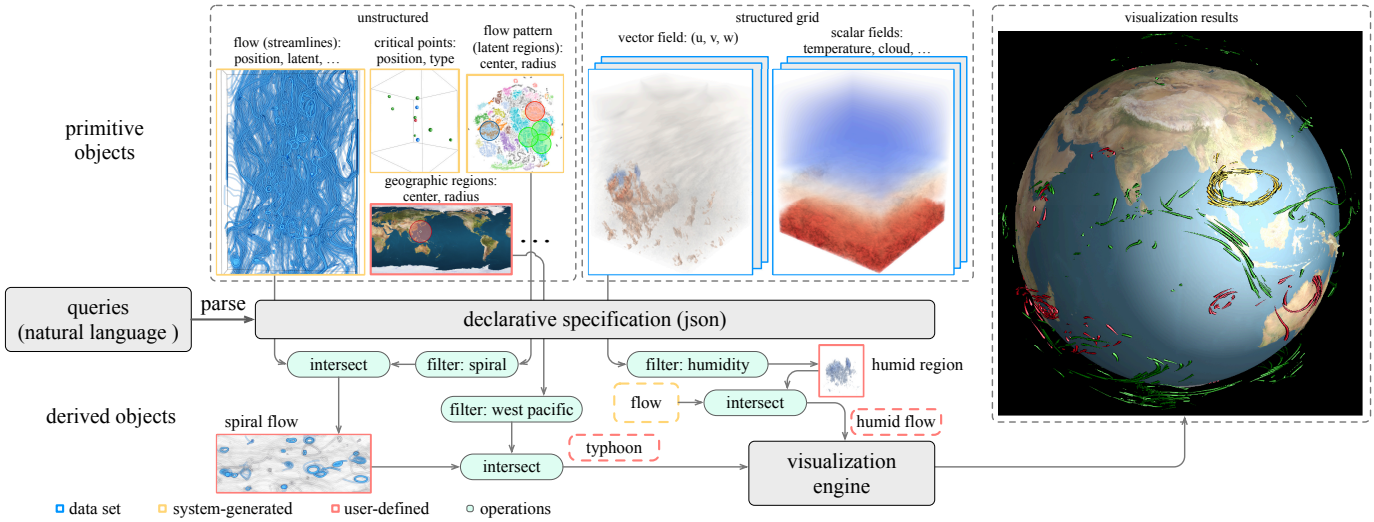


Fig. 2. An example of FlowNL workflow that translates the natural language queries into flow visualization results. The queries are parsed to form the declarative specification. The declarative specification specifies how to derive objects for visualization (□) from the existing primitive objects in the data set (□) and automatically generated data (□). The operations used to derive the objects are shown in the green boxes (□).

with both structured and unstructured data. Therefore, although the framework is similar, the problem formulation, the declarative language design, and the visualization engine of FlowNL are still unique.

Similar to existing declarative languages, FlowNL customizes visualization using dictionary-like specifications. But unlike other technique, FlowNL supports operations that are designed specifically for describing structures in flow fields. For example, using the left and right neighboring operations to extend a scalar feature along the flow.

3 Design Requirements

Target users and tasks. FlowNL aims at providing flexible interactions to explore flow field. It should allow users to conveniently specify structures of interest and observe their related flows. Toward this goal, it targets the common exploratory tasks for domain experts:

T1. Filtering. The experts want to identify structures in scalar attributes, features, spatial regions, and latent regions. For example, what are the flow in high temperature regions (i.e., heat transmission), and where are the spiral flows (e.g., vortices)?

T2. Compound queries. The experts want to specify structures fulfilling multiple criteria. For example, where are the upward flows with high humidity (i.e., evaporation), and where are the strong spiral flows (i.e., hurricane)?

T3. Transportation and connection discovery. The experts want to know how a structure evolves along the flow and how different structures are connected by the flow. For example, where the flow in high pollution regions goes to (i.e., pollution diffusion), and is there any pathway from the spirals to the sinks?

Targeting these tasks, FlowNL is design as a lightweight tool to utilize the information in data, but it may not extract complex features that requires sophisticated computation. Its primary target users are domain experts. They can use FlowNL to examine simulated data, observe extracted features, and verify scientific hypotheses with visualization. They can also generate animated visualization easily, which helps to communicate their research work with others. FlowNL may be useful in science popularization as well. Tutors can produce visualization to explain scientific phenomenons and the audience can interact to discover more. Guided by the analysis of tasks and users, we identify the design requirements as:

R1. Easy-to-use. The tool should support users with limited visualization background. Specifically, the users should be able to use natural language queries to identify the flow fulfilling specific criteria and create customized flow visualization easily.

R2. Predictable behavior. The tool should deliver trustable visualization results, in the sense that users can expect how the system will respond to their queries. This requires the rules to understand queries and determine the system's behavior to be easily explainable.

R3. Lagrangian view of flow. The tool should support the queries of Lagrangian flow behavior. For example, it should allow users to query where the flow in a region comes from and where it goes to.

R4. Application-agnostic. The tool should be able to support the common types of flow data (e.g., flow fields and their associated scalar fields), and detected features, regardless of the specific application.

4 Our Approach

Our FlowNL system is designed to translate natural language queries into flow visualization results. The queries specify what are the flow structure to visualize and how each structure should be visualized. Due to the difficulty to build direction connections between natural languages and flow visualizations, we first consider how to describe flow structures in a general way. Toward this end, we designed a declarative language that filters and combines the basic types of flow data (named primitive objects) to derive the other more complicated objects (i.e., flow structures). In this way, our goal becomes translating natural language queries into declarative specifications and rendering the flow structures according to the specifications.

Framework. Figure 2 illustrates an example of FlowNL workflow that translates the natural language queries into animated flow visualization. The translation first parses the queries to form the declarative specifications. The declarative specifications leverage the pre-existing knowledge from either the data set (□) or the data automatically generated by our system (□). The pre-existing knowledge is shown on the top. The declarative specifications specify the operations (□) to filter and combine the existing objects to derive the new ones (□). For example, the spiral pattern is generated as a filter of the primitive object "flow pattern" (vectors in deep latent space), and the spiral flow is produced by intersecting the spiral pattern and all the streamlines. Then, an object "typhoon" is generated by taking the intersection of the spiral flow and a geographic region "west Pacific Ocean". The declarative specification also decides which objects will be sent to the visualization engine. In this section, we will formally describe the objects, the declarative language, and the natural language processing.

4.1 Objects and Attributes

The objects abstract various kinds of data and flow structures, in accordance with design requirement **R4**. FlowNL considers two types of objects: *primitive objects* and *derived objects*. The primitive objects are basic ingredients that cannot be built from other objects, including all information provided by the data set or the system, and basic objects

defined by users (e.g., geographic regions). The derived objects are generated from primitive objects or other derived objects to describe the flow structures. In this section, we describe the concept of the objects and attributes. Please refer to Section 3 for an example of specification.

Primitive objects and attributes. Two types of primitive objects are considered in our current design: *structured grids* and *unstructured points*. The primitive objects of structured grids host the flow field and scalar fields as attributes. Fields of different resolutions can be considered as attributes of different grid objects. In this paper, we consider all fields to share the same resolution for simplicity. In this case, a single primitive object "grid" is created to include all the fields. Other flow objects usually consist of unstructured points, such as critical points, sample points on streamlines, and spatial regions. Each object consists of a set of points carrying various attributes. For example, all critical points can form an object, and each point may have its own position, type, and scale. With this definition, a specific type of critical point (e.g., sinks) can be a derived object from filtering the primitive object "critical points" by type.

The primitive objects encode the essential elements and their attributes in the data. Our system can support attributes of any dimensions, such as 1D scalar values, 2D and 3D positions, and even high-dimensional latent vectors from deep representation approaches. The primitive objects and their attributes are configurable using a meta file in JSON format. The dashed frames in Figure 2 show a typical example of primitive objects in a data set, which involves five types of data. *The first type is the field data from the data set*, as shown in the blue boxes. This includes a flow field and multiple scalar fields. *The second type is the sampled streamlines generated by the system*. The streamlines provide a Lagrangian view of the flow field, which is necessary to understand the flow behavior. To ensure the coverage of the field, an information-theoretic framework [56] is used to guide the sampling. Our system will also produce several attributes for filtering desired flow features. Currently, for each sample point, we record the position of that point and generate a latent vector to describe the flow pattern. The latent vector is produced by a deep autoencoder on the distance matrix among sample points. Users can use latent vectors from other representation approaches, as our system supports high dimensional attributes. FlowNL does not produce further attributes (e.g., vorticity) and rely on users to provide them as scalar fields if needed.

The other three types of data can all be considered as spheres in different spaces: the critical points are spheres in the 3D physical space, the flow patterns are spheres in the latent space, and the geographic regions are circles in the 2D geographic space. Note that we use the spheres to approximate regions in different spaces because this strategy can easily extend to arbitrary dimensions. For example, in our current implementation, the flow patterns are spheres in 128-dimensions. For an irregular region, multiple spheres can be used to approximate a single object. However, in the scenarios where high precision is required, users may adopt other approaches to approximate the regions (e.g., mask volumes).

Derived objects. The derived objects are generated from the primitive objects or other existing derived objects. FlowNL provides two schemes to derive objects. The first scheme is *filtering existing objects*. For example, an object of the saddle points can be created by filtering the critical points by their types, and an object corresponding to regions of high humidity can be created by filtering the grid points based on the humidity scalar field. The second type of object is generated by *combining existing objects* using a suite of operations provided by our system. For example, the hurricane is the intersection of the spiral flow and the geographic region "Gulf of Mexico". We will elaborate the derivation operations in the next subsection.

4.2 Declarative Language for Flow Visualization

The declarative language specifies visualization parameters for existing objects and operations to generate new objects. For object derivation, our system represents all objects as *subsets of elements of the respective primitive objects* and apply the following operations to derive new ones: filtering, mapping, union, intersection, difference, and neighboring. In this section, we will elaborate on the goals and rules of the operations,

```
{
   "task": "create object",  "name": "high humidity",        //filter the scalar field r
   "primitive": "grid",  "attribute": "r",                  //and select the grid points
   "operation": ">",  "value": "90%"                        //with top 10% of r values
}
{
   "task": "create object",  "name": "with high humidity",  //generete a derived object
   "operands": ["flow", "high humidity"],                   //with the streamline points
   "operation": "intersect"                                 //in high humidity region
}
{
   "task": "create object",  "name": "from high humidity",  //generate a derived object
   "operands": ["with high humidity"],                      //by extending the flow with
   "operation": "right neighbor"                            //high humidity along the flow direction
}
{
   "task": "visualize",  "name": "from high humidity",      //visualize the derived object
   "color": "drak red"                                      // with flow "from high humidity"
}
```

Fig. 3. Sample declarative specifications generated by the query "show the flow from high humidity region". The four declarative specification corresponds to the tasks to filter the grid by humidity, identify the streamline segments in the high humidity region, extend the segments along the flow direction, and visualize the segments.

but only briefly introduce the syntax with examples, as the users are supposed to interact with the system using natural language instead of the declarative language.

Filtering. The filtering selects a subset of elements from an object to form a new one based on an attribute. Note that the attribute can be 1D or high-dimensional. The high-dimensional filter is represented by a series of spheres. Each sphere is encoded by its center (i.e., high-dimensional vector) and a radius.

Mapping. The mapping converts a derived object from one primitive to another using a specified attribute. For example, streamlines can be mapped to the nearest grid points based on their positions. In this way, a derived object from the primitive "flow" can be converted into an object of the primitive "grid".

Union, intersection, and difference. These operations apply respective set operations to combine the elements of two objects. When both the two objects are derived from the same primitive objects, the operations are performed directly on the indices of elements.

Operations between different primitive objects. When the two objects are derived from different primitive objects (for example, humid regions and spiral streamlines), a mapping will be performed to map the elements in the second operand to the elements in the first operand. For example, the statement "humid regions containing spiral streamlines" will map the spiral streamline segments into grid points and compute the intersection on grid points; while "spiral streamlines in humid regions" will map the grid points with high humidity to streamlines and perform the intersection on streamline points. In short, the resulting object shares the same primitive object as the first operand. Therefore, the first statement produces a region while the second one produces a series of streamline segments.

Neighboring. The neighboring operation extends the spatial coverage of an object to provide more contextual information. For grid points, the neighboring operation includes the neighboring grid points of the current ones, which is similar to the dilation in image processing. For spatial regions, the neighboring operation simply increases the radius of each circular region. For streamline segments, the neighboring operation expands the segments along the streamlines.

In accordance with design requirement **R3**, FlowNL facilitates the understanding of the flow field from the Lagrangian perspective. We further introduce the *right neighboring* and *left neighboring* to expand the segments along the flow direction and the opposite, respectively. These operations are particularly useful for tracking the origin-destination relationships in flow fields. Note that the objects that are not derived from streamlines do not support the these two operations. In this case, the system will automatically map the objects to streamlines, apply the operation, and map the object back to its original primitive.

Query formula. The query formula is designed to specify compound operations to simplify declarative specifications. It also serves as a brief description of a derived object and a convenient interface for

users to formally define an object. The operations supported by the query formula are: union (|), intersection (&), difference (−), neighboring ($\mathtt{N}(\cdot)$), left neighboring ($\mathtt{L}(\cdot)$), and right neighboring ($\mathtt{R}(\cdot)$). We do not support filtering in the query formula for simplicity, as filtering is more convenient using our filtering widgets. For example, the query "show the flow from high humidity region" in Figure 3 can be written as "$\mathtt{R}([flow]\&[highhumidity])$" in the query formula.

Visualization. The flow-related objects (i.e., derived from streamlines) are used as seeding candidates to generate particles with a small noise applied. To avoid a particle diverging from the original object, the flow direction at the particle is compared to the direction on the original streamline. When the difference between the direction is too large, the particle will be directly recycled. A declarative specification can also specify the color and the density of particles related to an object. It can specify a scalar field for color mapping as well. The other objects are visualized as point clouds. In this way, the statement "show the flow of high humidity" will create particles in the high humidity regions, while "show the region of high humidity" will create a point cloud covering the high humidity regions.

4.3 Natural Language Processing

The declarative language may still be difficult to use for domain experts without programming experience. To further enhance the usability, the FlowNL architecture is built with a natural language interface (NLI) to translate natural language queries from users to declarative language. The NLI consists of: a *grammar parser* that identifies the tasks, objects, and their relationships from the natural language; a *derivation mechanism* that generates derived concepts from existing ones; and additional features such as *ambiguity resolving* and *auto-completion*. Note that we choose semantic parser for it is easier to explain and manipulate, in response to design requirement **R2**.

Grammar parser. The parser of FlowNL translates a user query into declarative specifications, including tasks, objects, object relations, and visualization styles. Specifically, FlowNL uses a context-free grammar to parse the queries and form a parser tree. Since a query in natural language is likely to specify multiple tasks at the same time, the parser tree usually contains multiple layers, where each node corresponds to a declarative specification. The fundamental grammar rules of FlowNL are as follows:

$$\begin{aligned}
&\textbf{Rule 1}: \langle Query\rangle \longrightarrow \langle TaskType\rangle\langle Object\rangle\langle Style\rangle\\
&\textbf{Rule 2}: \langle Object\rangle \longrightarrow \langle Object\rangle\langle ObjectRelations\rangle\\
&\textbf{Rule 3}: \langle Object\rangle \longrightarrow \langle Object\rangle\langle Object\rangle\langle ObjectRelations\rangle\\
&\textbf{Rule 4}: \langle Object\rangle \longrightarrow \langle Attribute\rangle\langle Object\rangle\langle Operator\rangle\langle Value\rangle
\end{aligned}$$

An example of the parsing procedure. Figure 4 shows an example of the parsing procedure for the query "Draw the spiral flow from Pacific with velocity magnitude over 32 in dark blue". The word "draw" identifies the query to be a "visualize" task using Rule 1, although the $\langle Object\rangle$ and $\langle Style\rangle$ are not identified yet. The phrase "spiral flow from Pacific" matches Rule 3 and produces the specification (a). The phrase "velocity magnitude over 32" matches Rule 4 and produces the specification (b). With the objects corresponding to the specification (a) and (b), the third object can be created with the specification (c) using Rule 3. Given the third object, the parser can determine the $\langle Object\rangle$ of the entire query and apply Rule 1 to generate the specification (d). In this example, the $\langle Object\rangle$ of the entire query is recursively identified from the primitive objects and their attributes. This procedure allows intermediate objects to be derived during the query. It distinguishes our FlowNL from most of the existing NLI approaches for tabular data, where entities are usually predefined by the columns of tables.

Object relations. $\langle ObjectRelations\rangle$ specifies how the objects relate to each other, which is used to generate the operations in the declarative specifications. Other than the set operations such as $\langle Union\rangle$, $\langle Intersection\rangle$, and $\langle Difference\rangle$, $\langle ObjectRelations\rangle$ further introduces the following spatial relations: $\langle From\rangle$, $\langle To\rangle$, $\langle FromTo\rangle$, $\langle Between\rangle$, $\langle In\rangle$, $\langle Near\rangle$, and $\langle RelatedTo\rangle$.

The spatial relations in $\langle ObjectRelations\rangle$ enable the analysis of Lagrangian flow behaviors. These relations extend the objects along the flow using the neighboring operations. The definitions of $\langle From\rangle$, $\langle To\rangle$, $\langle FromTo\rangle$, and $\langle Between\rangle$ are relatively straightforward, and $\langle In\rangle$, $\langle Near\rangle$, and $\langle RelatedTo\rangle$ specify neighboring regions of growing sizes. $\langle In\rangle$ specifies the flow inside the object, $\langle Near\rangle$ specifies a small neighborhood of the object by extending the object along the flow, while $\langle RelatedTo\rangle$ specifies the entire region containing the streamlines passing through the object. Formally, let $\mathbf{O}$ be an object, $\langle From\rangle$ indicates $\mathtt{R}(\mathbf{O})$, $\langle To\rangle$ indicates $\mathtt{L}(\mathbf{O})$, $\langle FromTo\rangle$ indicates $\mathtt{R}(\mathbf{O}_a)\cap\mathtt{L}(\mathbf{O}_b)$, $\langle Between\rangle$ indicates $(\mathtt{R}(\mathbf{O}_a)\cap\mathtt{L}(\mathbf{O}_b))\cup(\mathtt{L}(\mathbf{O}_a)\cap\mathtt{R}(\mathbf{O}_b))$, $\langle In\rangle$ indicates $\mathbf{O}$, $\langle Near\rangle$ indicates $\mathtt{N}(\mathbf{O})$, and $\langle RelatedTo\rangle$ indicates $\mathtt{N}+(\mathbf{O})$, where "+" denotes an extended neighborhood.

Task identification. FlowNL uses keyword classification for task recognition. The parser will first match the words in a query sentence with the keywords in a dictionary, which lists potential keywords of our tasks. For example, "show" and "draw" will be classified as a query of the "visualize" task. If none of the keywords is matched, the synonyms corresponding to these keywords will be retrieved automatically based on word similarity. In our implementation, the similarity between words is calculated by the Wu-Palmer similarity function [55]. This similarity function returns a similarity score based on the depth of the two words in WordNet classification [30] and the depth of their least common substrate (LCS) of the most specific ancestor node.

Note that the task of some verbs may rely on the content of the sentence. For example, "set" and "change" may mean to adjust the color of an object (e.g., "change the color of A to red"), corresponding to the *color* task. In other case, they may mean to adjust the threshold of a filter (i.e., "set the wind speed of hurricane over 40"), corresponding to the *adjust* task. For this kind of ambiguous keywords, the system will further search the entire sentences for the actual task.

Object identification. An object in a natural query may be the name of an attribute, a primitive object, an existing derived object, and even the name of an object to be derived. We use the part-of-speech tagging (POS) tagging to recognize unknown objects and N-grams to check the existing objects, which is similar to the previous NLI approaches [42, 44, 59]. But unlike these approaches, we do not allow approximiate matching as scientific concepts and terms often carry exact meanings. For example, "typhoon" and "hurricane" shares a large similarity value (larger than 0.9) using the Wu-Palmer similarity function [55], but they may refer to difference concepts for researchers in atmospheric science. Therefore, instead of auto-correction, we rely on users to provide the exact terms for query.

To reduce the user effort, we allow users to provide aliases for the objects. These aliases will be used in the N-gram matching in addition to the object names. Users do not need to define an alias before using it. Instead, they can provide the definition in a dialog using our derivation mechanism, which will be explained in the following paragraph.

Derivation mechanism. We introduce a derivation mechanism to resolve the undefined objects or alias. The undefined terms are identified as unrecognized noun phrase by tracking the POS tagging produced and the dependency tree using the Stanford CoreNLP toolkit [29] and SpaCy [15]. Figure 5 illustrates an example of derivation. When a user ask the system to show an unknown object "hurricane", the system will ask the user for the definition. The user may then explain that "hurricanes are spirals with wind speed larger than 32 meters per second near Gulf of Mexico". Upon receiving this explanation, the parser will parse the sentence, generate the declarative specification, and send the specification to the visualization engine to derive the new object "hurricane". With the object created, the natural language parser will recursively process the original query "show hurricane" and request the visualization engine to visualize the hurricane.

We rely on users to provide the definitions of unrecognized objects based on the existing ones, instead of querying knowledge bases such as WolframAlpha [1] and WordNet's synsets [30] to obtain the definition automatically. This avoids the ambiguity of scientific concepts, which could be domain-specific. For example, the wind speed to characterize a hurricane may vary across different research areas. Indeed, the Saffir-Simpson hurricane scale categorize hurricanes into five types based

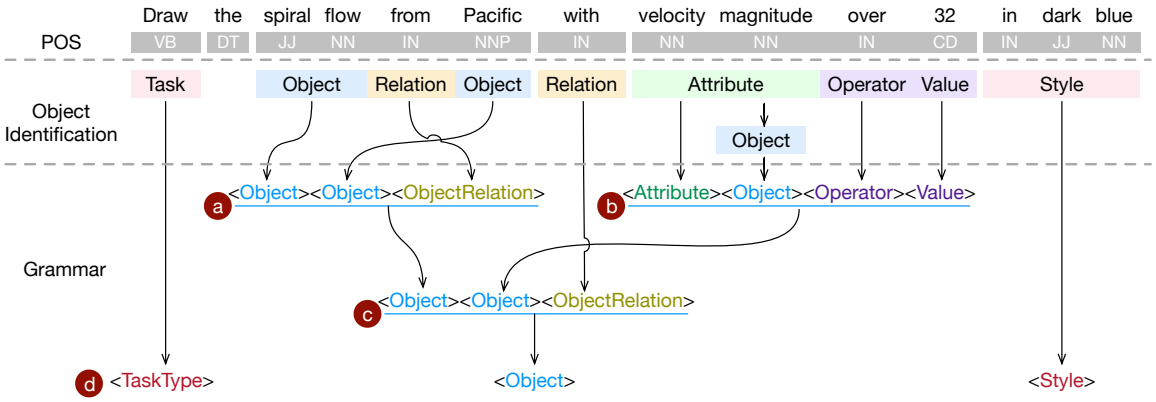


Fig. 4. An example of parsing procedure. From top to bottom, this figure shows the natural language query, the part-of-speech (POS) tag, the object identification, and the parsing tree using grammar rules. The rules labeled by the numbers are used to produce the declarative specifications.

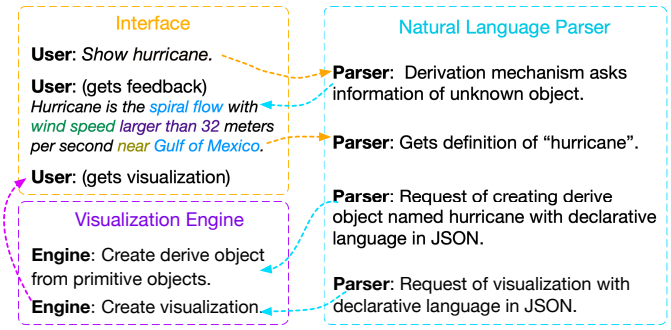
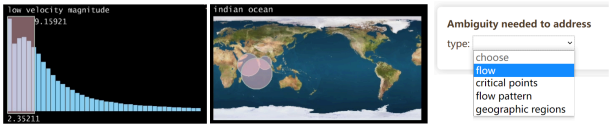


Fig. 5. An example derivation workflow to define "hurricane".



(a) (b) (c)

Fig. 6. Resolving ambiguity in FlowNL. (a) shows the widget to specify the value range of "low velocity magnitude" . (b) shows the widget to specify the region of "Indian Ocean". (c) shows a box to specify the primitive for an attribute named "type".

on the wind speed. Additionally, the term "hurricane" may even be used to describe general large swirling storms without specifying the geographic location.

Resolving ambiguity. Two types of ambiguity may appear in natural language queries: *semantic ambiguity* and *quantitative ambiguity*. The semantic ambiguity occurs when objects or their attributes share the same names. For example, the term "spiral" may refer to a type of critical points, or a flow pattern, leading to semantic ambiguity. In this case, FlowNL will make a best guess based on a least recently used (LRU) strategy, assuming that an object will not be specified in multiple consecutive queries. But we should note that FlowNL does not completely rely on this strategy to determine the object referred by an ambiguous term. Instead, FlowNL will expose the ambiguity of the phrase, present all potential objects to users, and allow them to correct the default choice of object, as shown in Figure 6(c).

The quantitative ambiguity refers to the vagueness of describing quantities in natural languages. For example, in the phrase "the flow with low velocity magnitude", the word "low" implies a filter should be applied but it does not specify the exact threshold for filtering. In this case, FlowNL will use a default threshold (e.g., "high" indicates "top 5%") and bring up an interactive widget to resolve the ambiguity. For scalar value, a histogram of the corresponding attribute will be displayed for selecting a value range, as shown in Figure 6 (a). For 2D vectors (e.g., geographic regions), a 2D plot will be displayed for users to specify circular regions, as shown in Figure 6 (b). Specifying regions in higher dimensional space (e.g., latent spaces) will be difficult, which is not supported in our current system. A potential solution is to use dimension reduction techniques to embed the targeted space into 2D space for selection.

Auto-completion. Auto-completion reduces users' effort to type in their queries, and also provides hints to users about the tasks and queries supported by the system. FlowNL implement this mechanism through fuzzy matching and prefix matching. We design a suite of template sentences to remind the users of the typical queries supported by our system. Before users enter any information, the input box will show the template queries in a default order, as shown in Figure 1 (e). Once users start typing, the system will keep updating the edit distances from the templates to the user input. The edit distance measures the minimum number of operations to convert the current input to the template queries, which reflects user's intention. We use the edit distance for selecting the candidate templates to display as suggestions.

4.4 Interface

The interface of FlowNL consists of: *query input box*, *dialog box*, *object and attribute table*, and *flow visualization*, as shown in Figure 1.

The query input box is shown on the top of the interface. It takes the natural language queries from users and sends the queries to the back-end natural language parser. During typing, it will keep tracking the key words including the objects, attributes, and colors. It will also match the partially completed queries with templates and display the top matching templates as suggestions, as shown in Figure 1 (e). The dialog box is used to define unknown terms during conversation, as shown in Figure 1 (b). If the definition contains further unknown terms, the dialog will be expanded to define the newly appearing terms.

The flow visualization shows the flow-related objects as streamlets and the other objects as point-clouds, as shown in Figure 1 (f). It also displays the interactive widgets to edit filters for 1D/2D attributes. The flow visualization is rendered on the back-end visualization engine, and sent to the front-end web interface as a video stream.

The object and attribute table shows the derived objects and attributes associated with each primitive object. The primitive objects are shown as the table headers in bold font. When the derived objects tab is active, each cell shows a derived object, as shown in Figure 1 (d). If an object is visualized, a small glyph will appear to its left. The glyph contains a color legend at the center and a circular slider at the outer ring. The color legend indicates the color of the object or the color map used for that object. The slider can be used to adjust the "weight" of the object. For a flow-related object, the weight indicates the density of particles assigned to the object. For an object of point-clouds, the weight is used as the opacity of the points. By clicking the object name, its

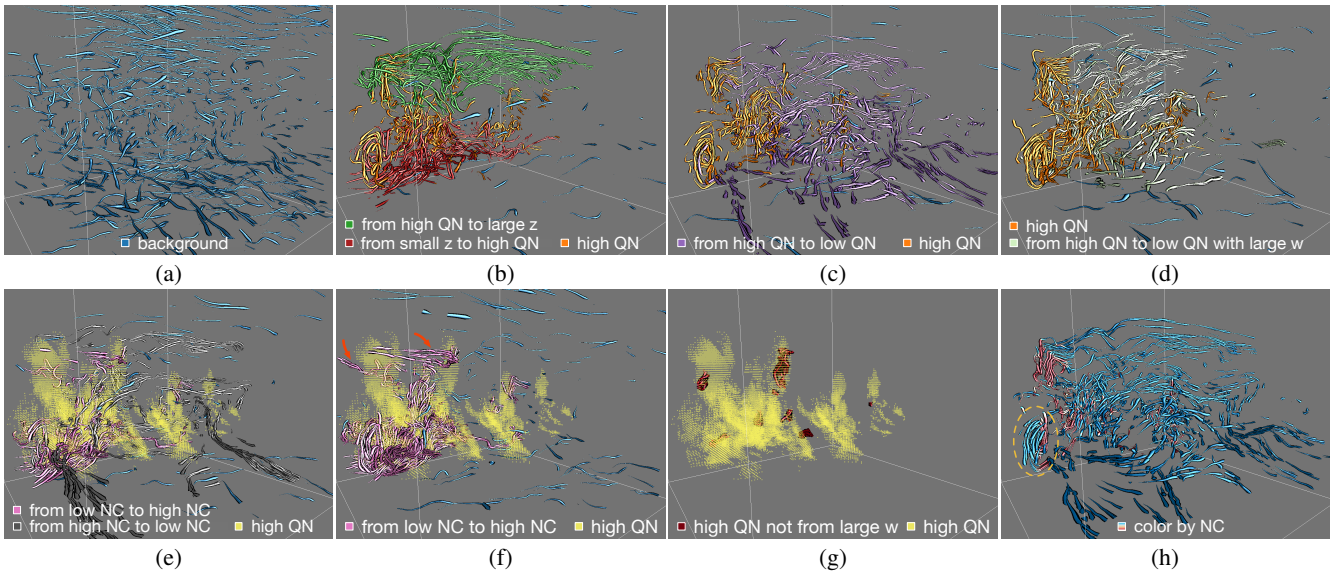


Fig. 7. The key visualization results from a domain expert's exploration using the BOMEX data. The blue streamlets correspond to the background flow, which are uniformly spawned in the entire domain. The variables "QN", "NC", "z", and "w" denote "non-precipitating condensate", "could water number concentration", "height", and "vertical velocity", respectively.

corresponding query formula will be displayed, as shown in Figure 1 (c). Users may edit the query formula to change its definition. If the object is a 1D or 2D filter, a selection widget will be displayed for users to edit the filter parameters. In Figure 1, a filter object "high w" is clicked and the histogram of the attribute "w" is displayed at the bottom left corner of the flow visualization. Users may brush the histogram to determine the value range of "high w".

5 Evaluation

We evaluate FlowNL empirically with a domain expert in cloud physics and a tutor in science popularization. Please refer to the supplementary video for the exploration of two case studies. We also conduct a formal user study to examine the learnability and usability.

5.1 Case Study by Domain Expert: BOMEX data set

We team up with an expert from Brookhaven National Laboratory to examine the effectiveness of FlowNL. The expert is a research scientist with more than 10 years of experience in cloud physics study. His research interest is ice nucleation, remote sensing, aerosol-cloud interactions, and cloud simulation. The expert provided the BOMEX data and used FlowNL to explore this data. The BOMEX data simulates the atmospheric shallow cumulus convection in a domain of $3175 \times 3175 \times 3980$ meters. This data set includes one flow field and fourteen scalar fields. The data contains a large cloud, and the expert would like to visualize the formation of this cloud by tracking the cloud droplet activation and evaporation. In this session, we present his exploration and reproduce the key visualization results, as shown in Figure 7.

The expert first examined the overall flow pattern with the randomly sampled particles, as shown in Figure 7 (a). In the upper layer, the flow is mostly laminar and moves at a fast speed. In the lower layer close to the ground, the flow becomes more turbulent and the flow directions are more diverse. In the middle layer, where the cloud resides, the flow pattern is more complicated and the streamlets are shorter as the flow moves slower in this region. Therefore, the randomly spawned particles are less effective in forming a continuous pattern to describe the flow behavior in this region.

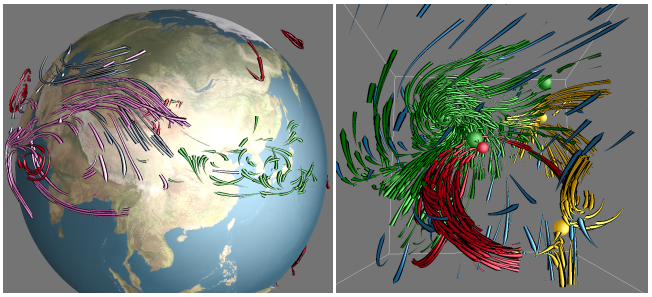
To better observe the flow pattern in the cloud regions, the expert queried "show the flow of high QN", where "QN" is the "non-precipitating condensate" including water and ice. He explained that "high QN" can be considered as the core of the cloud. The "flow of high QN" is visualized in orange in Figure 7 (b). To further examine the interaction between the cloud and the layers of flows close to it, the expert queried "show the flow from high QN to large z" (in green) and "show the flow from small z to high QN" (in red), where "z" is the height. He explained that the red flow and the green flow revealed how the vapor entered and escaped from the cloud core, respectively.

However, the interactions between layers of flow may not reveal the complete paths of vapors. Therefore, the expert hid the red and green flow and added the purple flow using a query "show the flow from high QN to low QN", as shown in Figure 7 (c). He commented that this showed a clearer pattern of the vapor transportation path. The particles move either toward the east (from the left to the right in the screen) or the southeast (from the left to the bottom right in the screen). The expert explained that this indicated that the cloud was shearing.

Next, the expert wanted to quantify the numbers of particles moving upward and downward, respectively. But this is not supported in our current implementation. Therefore, he decided to separate the upward flows, as this was usually less common. He removed the purple flow to reduce occlusion and added the upward flow by querying "show the flow from high QN to low QN with positive w", where "w" is the vertical velocity. The corresponding flow was shown in light green in Figure 7 (d). The upward flow leaving the cloud core mostly move eastward. This indicates that cloud dilution, arising from the mixing between the cloud entity and the environmental air, occurs in the downwind region due to the wind shear.

Then the expert used the attribute "cloud water number concentration" (NC) to guide the exploration. He explained that NC was a better indicator of the boundary of the cloud, as this attribute is stable inside the cloud and mostly zero outside the cloud. In contrast, QN change smoothly from the boundary to the core of the cloud. The expert used a yellow point cloud to indicate the spatial coverage of the core of cloud by querying "show the grid of high QN in light yellow". This avoided the distraction from the orange streamlets. He then added the inward and outward flows by querying "show the flow from low NC to high NC" (light purple) and "show the flow from high NC to low NC" (gray), as shown in Figure 7 (e). He commented that the flows generated using NC better demonstrated the interactions near the cloud boundary.

The expert was particularly interested in the light purple flow (from low NC to high NC), as this flow supported the cloud. Therefore, he removed the gray flow for better observation of the light purple one, as shown in Figure 7 (f). Most of the light purple flow resides at the bottom of the core of cloud, supporting the base of the cloud. But the expert also found two branches of flow entering the cloud on the top, as highlighted by the red arrows. This may indicate the cloud droplet



(a) (b)

Fig. 8. Flow visualization results using additional data sets. (a) shows the result using ECMWF data set. (b) shows the result using five critical points data set.

formation at cloud edge due to entrainment and mixing. The expert was curious about whether there was any other entry point from non-upward flow. He queried "show the flow from low NC to high NC that is not from large w". The query result is visualized by the red streamlets in Figure 7 (g), which reveals all entry points from the non-upward flow.

Finally, the expert examined the evaporation process based on NC. He brought back the orange and purple flows in Figure 7 (c) and queried "color the flows by NC", as shown in Figure 7 (h). The flow shows a clear boundary between the red and blue regions, indicating the boundary of the cloud. The expert found that a vortex appeared at the subsiding shell, as highlighted by the dashed orange circle. He commented that this vortex clearly showed the entrainment (environmental air flows into the cloud entity) and detrainment (part of cloud entity flows into the nearby environment) processes.

Overall, the expert was satisfied with the exploration using FlowNL. He commented that "FlowNL is a powerful tool to visualize the instantaneous 3D fluid motion. It helps to illustrate complex processes in a turbulence environment (e.g., atmospheric clouds), which can benefit education and research." He also stated that "I will recommend FlowNL to my colleagues and I will be interested in using the future versions. Particularly, it will be even more helpful if it can support quantification of the particles with different criteria and plot the statistics in nice visualization charts."

5.2 Additional Case Studies

European Centre for Medium-Range Weather Forecasts (ECMWF) data set. A tutor in science popularization used FlowNL to generate an animation for scientific storytelling. She aimed at explaining the vapor transportation to China. A key visualization frame is shown in Figure 8 (a). She first queried "show the hurricane" and explained to the system that hurricane is "spiral flow". She showed the typhoon as the hurricane in west Pacific in green. She colored the hurricanes that were not typhoon in red. She then identified two vapor transportation paths from the Mediterranean Sea to China and from the Indian Ocean to China. She was satisfied with the video produced.

Five critical points. We used FlowNL to visualize this data set, as shown in Figure 8 (b). We first showed the saddles, sources, and spirals in yellow, red, and green, respectively. Then we showed the flow related to the saddles, the flow from the sources, and the flow from the spirals in the respective colors as the critical points. Note that there are several spiral saddles in this data set. We queried "show the flow related to the saddles but not from the spirals" to avoid overlaps.

5.3 User Study

We conduct a formal user study to evaluate the effectiveness, usability, and learnability of our FlowNL. Through this study, we wanted to evaluate how much learning effort it costs for a user to manipulate the tool, whether a user can form correct queries for given tasks, whether a user is confident in the meaning of the resulting visualization, and how well FlowNL responds to the user queries.

Participants and experiment setup. We recruited eight unpaid participants with either computer science or scientific domain background. Two participants have PhD degrees, four have master degrees, and the other two have bachelor degrees. In terms of academic background, five participants have mechanical engineering, physics, or atmosphere research experience, three participants have research or working experience in interactive techniques (including one with atmosphere background), and the other participant's research area is database. Four participants reported experience with geographic information systems, ParaView, or similar tools, but none of them have experience with interactive flow visualization system or natural language interface. The participants interacted with the system on a 32-inch 4K screen where FlowNL interface occupied an area of 2600×1400 pixels.

Procedure. The participants started with a demonstration video and a brief introduction (approximately 15-20 minutes). The introduction included the natural language query, query formula, interface of FlowNL, and basic knowledge about the ECMWF data set, which was used in this study. After the introduction, they could freely experiment with FlowNL to explore the ECMWF data set. They could ask questions regarding the usage of any functions of FlowNL or the data set. They were informed to take as long as they want until they feel comfortable to perform the tasks. Each of the participants was required to perform 12 tasks using the ECMWF data on a different date. Their interactions were recorded as a video by a screen capture software. They were informed that the timing would not be used to measure their performance, and they were encouraged to experiment different queries for the same task. Finally, the participants were required to answer the ten questions in SUS questionnaire [4] and provide additional comments. Please refer to the supplemental materials for the introduction and questionnaire documents.

Tasks. The twelve tasks covered different types of queries, including creating derived objects, verifying query formula, hiding an object, adjusting the density of particles for an object, extending an existing object along the flow, adjusting the value range of a filter, and changing the color of objects. Five tasks were related to the object creation, including objects related to attributes, geographic regions, flow patterns, and compound relations.

	P1	P2	P3	P4	P5	P6	P7	P8	avg.
time (min)	42	41	32	29	23	19	27	17	28.8
# first	11	9	11	16	12	12	12	14	12.1
# total	39	55	39	40	32	18	22	40	35.6
# accepted	33	51	30	38	31	18	19	39	32.4
# QF	4	0	5	0	4	2	5	1	2.6

Table 1. The task performance. # first shows the numbers of queries to finish the tasks for the first time, # total shows the total numbers of queries performed, # accepted shows the numbers of accepted queries, # QF shows the numbers of query formula used.

Task performance. All the participants performed all tasks successfully, although using different number of queries. As most of the participants were willing to experiment with alternative queries to explore more, we report both the number of queries for them to complete the tasks for the first time, and the total number of queries they performed. Table 1 summarizes the task performance.

In terms of the first time completion, the participants used 12.1 queries to perform the eight tasks requiring at least one query. We did not count the other four tasks as they could be performed with mouse operations instead of queries. For the eight tasks being counted, the possible minimum number of queries is 9, as one of the task requires at least two queries to perform. One participant (P2) used exactly 9 queries to finish the tasks and five participants used 12 queries or less to finish. P4 used 16 queries to finish the tasks, which was the most among all participants. By checking the recorded video, we found that P4 used 6 queries for Task 5, which asked the participants to create a flow-related object fulfilling three criteria. P4 used multiple queries to create additional objects, leading to extra number of queries.

The exploration time and the total number of queries varied across participants. The participants took 28.8 minutes to perform the tasks on average, ranging from 17 to 42 minutes. On average, they performed 35.6 queries with 32.4 queries accepted, leading to an acceptance rate of

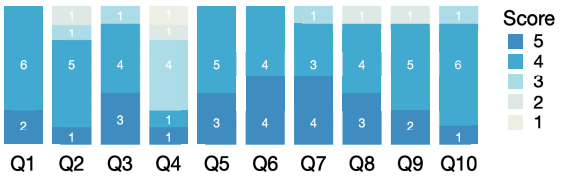


Fig. 9. Participant responses for the ten SUS questions. The higher scores are placed closer to the bottom.

90.9%. Most of the participants performed much more queries than the minimum required to finish the tasks. We found that the acceptance rate might not correlate with the exploration time. The three participants who spent the least amount of time (17, 18, and 23 minutes) actually had high acceptance rate (39/40, 18/18, and 31/32, respectively). Please refer to Section 1 in Appendix for the details of rejected cases.

Most of the participants were willing to experiment with the query formula. Four participants used the query formula to create or edit an object for more than four times, even if the query formula was not necessary to perform the tasks. Only two participants did not use the query formula at all during the exploration. To our surprise, the five participants with scientific domain background were not reluctant to try the query formula. All of them used the query formula for at least once, including one used it for five times and two used it for four times.

SUS Scores. We evaluated the learnability and usability of FlowNL using the System Usability Scale (SUS) [4]. FlowNL received an overall score of 76.6 out of 100, placing it between "good" and "excellent" in SUS rating. Figure 9 shows the detailed survey results. Note that, since SUS alternates the tone of each item, we reverse the scores for negatively phrased items back to positive scores before the analysis.

Three questions are regarding the learnability: Q4 ("I need technical support to use this system"), Q7 ("I think most people can learn this system quickly") and Q10 ("I think there are a lot of things to learn before I can start using this system"). While Q4 received the lowest score (3.0) among all the questions, both Q7 (4.4) and Q10 (4.0) received relatively high scores of 4.4 and 4.0, respectively. It seemed that most participants believed that one could learn FlowNL in fairly short amount of time without much background, but they felt that the technical support was necessary during the learning stage.

The other questions are related to the usability. All questions except Q2 received scores of 4 or above, indicating the participants agreed that the system functioned smoothly without much irregularities. Q2 ("I find this system to be more complicated than it should be") received the lowest score of 3.75. One participant explained that it was common to analyze planar flows in atmospherical research, where static visualization was adequate.

User behaviors. We have two main observations in the user study. First, *mouse versus natural language queries.* The participants still preferred mouse operations for some tasks. For example, we found that only three participants used the natural language for Task 7 (adjust the density of particles) and 8 (hide a specific object). Two participants mentioned that they would use natural language if voice queries were supported. By analyzing the recorded interactions, we also found that the participants were accustomed to mouse operations. For example, one participant moved the mouse to an object after executing the natural language query to hide that object. We believe this is also related to the use of desktop in our experiment. In an immersive environment, the participants may lean to the natural language queries.

Second, *ambiguity in natural language*. For example, four participants expected that "the flow passing through China and Japan" indicated an intersection between the flow through China and the flow through Japan, while the other four participants corresponded this query to the flow through China or through Japan. We also found that this might relate to the underlying geographic locations, as most of the participants believed that "the flow passing through China and Mexico" indicated a union. Under the design requirement **R2**, we always use the intersection for "and" and union for "or". We explained this rule to the participants so that the behavior of FlowNL can be predictable.

5.4 Limitations

Unsteady flow fields. The current evaluation only examines the effectiveness of FlowNL using steady vector fields and streamlines. This may be acceptable when the change of flow is much slower than the movement of particles. However, for general applications, unsteady flows and pathlines must be studied to understand the Lagrangian behavior. This is not fully supported in the current version of FlowNL. Our data engine can naturally incorporate pathlines as a primitive with 4D points, and the set and neighboring operations can apply to the pathline objects. However, the visualization engine needs further development to filter objects by time during animation, and the natural language parser needs an extension to support the navigation of time. Furthermore, the scale of unsteady fields may require more powerful devices or more advanced algorithms to manipulate objects in real-time.

Complex features. We should note that FlowNL cannot derive complex features directly from the raw flow fields. For example, vortex extraction [13] may be computed by reference frame optimization [12], from trajectories [48], from surfaces [11], etc. These computations, while accurate, are application-specific, and they cannot be performed with FlowNL. Under design requirement **R4**, FlowNL provides a general data interface for users to provide their features.

Meanwhile, FlowNL can incorporate deep representations as attributes, which is useful with the emerging deep learning-based techniques in scientific visualization. Currently, FlowNL uses latent encoding as an attribute of streamlines to describe their shapes. We manually select typical encoding from the embedding space as flow patterns. For example, "spiral" corresponds to the streamline segments of a circular shape. Compared to the specifically designed algorithms, our encoding is less accurate and only reveals patterns of individual streamlines. But it serves as an immediately available tool for rough explorations.

Visualization and interaction techniques. FlowNL visualizes various kinds of objects in two forms: streamlet animation and point clouds. However, this may not reveal complex structures concisely. More involved techniques, such as integral surfaces, should be considered. Additionally, the scalar features may be rendered more precisely using direct volume rendering or isosurface rendering as well. The natural language parser and declarative grammar should be extended to specify visualization styles and parameters. In terms of interaction, FlowNL uses textual input, which limits its usage. Voice may be more applicable in immersive environments and enhances multimodal interactions, as voice interaction is hands-free.

Evaluation. FlowNL is evaluated by two experts through empirical evaluation, and five participants with domain backgrounds in the user study. Deploying the system to a public platform may better evaluate its performance in different applications and scenarios. For now, we provide an analysis about the supported users and tasks in Section 3.

6 Conclusions and Future Work

We propose FlowNL, a natural language interface for exploratory flow visualization. FlowNL integrates a natural language parser, a declarative language designed for flow visualization, and a flow visualization engine. It supports queries of flow structures related to various kinds of scalar and flow features. We evaluate FlowNL using multiple case studies with domain experts and a user study.

While our current implementation emphasizes predictable responses for queries, we would like to explore more sophisticated approaches in understanding the fuzzy natural language queries. For example, by deploying FlowNL on supercomputers, we may collect more queries from a broader spectrum of users to train deep translators. Knowledge graphs may be used to derive unknown concepts as well. We would also like to support unsteady flows, more involved visualization techniques, and multimodal interactions, as discussed in Section 5.4.

Acknowledgments

This research was supported in part by the National Natural Science Foundation of China through grants 61902446, 62172456, 91937302, National Key R&D Program of China through grant 2021YFB0300103, and National Windtunnel Project. The authors would like to thank Dr. Fan Yang for his insightful suggestions and case study.

REFERENCES

[1] WolframAlpha. https://www.wolframalpha.com/.

[2] R. Bader, M. Sprenger, N. Ban, S. Rüdisühli, C. Schär, and T. Günther. Extraction and visual analysis of potential vorticity banners around the alps. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):259–269, 2019.

[3] M. Bostock, V. Ogievetsky, and J. Heer. D^3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[4] J. Brooke. SUS: A quick and dirty usability scale. *Usability Evaluation in Industry*, 189(3), 1996.

[5] S. Bruckner and M. Groller. VolumeShop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization Conference*, pages 671–678, 2005.

[6] R. Bujack, J. Kasten, I. Hotz, G. Scheuermann, and E. Hitzer. Moment invariants for 3D flow fields via normalization. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 9–16, 2015.

[7] H. Choi, W. Choi, T. M. Quan, D. G. C. Hildebrand, H. Pfister, and W.-K. Jeong. Vivaldi: A domain-specific language for volume processing and visualization on distributed heterogeneous systems. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2407–2416, 2014.

[8] W. Cui, X. Zhang, Y. Wang, H. Huang, B. Chen, L. Fang, H. Zhang, J.-G. Lou, and D. Zhang. Text-to-Viz: Automatic Generation of Infographics from Proportion-Related Natural Language Statements. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):906–916, 2020.

[9] K. Dhamdhere, K. S. McCurley, R. Nahmias, M. Sundararajan, and Q. Yan. Analyza: Exploring Data with Conversation. In *Proceedings of International Conference on Intelligent User Interfaces*, pages 493–504, 2017.

[10] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios. DataTone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of Annual ACM Symposium on User Interface Software and Technology*, pages 489–500, 2015.

[11] C. Garth, X. Tricoche, T. Salzbrunn, T. Bobach, and G. Scheuermann. Surface techniques for vortex visualization. In *Symposium on Visualization*, volume 4, pages 155–164, 2004.

[12] T. Günther, M. Gross, and H. Theisel. Generic objective vortices for flow visualization. *ACM Transactions on Graphics*, 36(4):1–11, 2017.

[13] T. Günther and H. Theisel. The state of the art in vortex extraction. In *Computer Graphics Forum*, volume 37, pages 149–173, 2018.

[14] J. Han, J. Tao, and C. Wang. FlowNet: A deep learning framework for clustering and selection of streamlines and stream surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 26(4):1732–1744, 2018.

[15] M. Honnibal and I. Montani. spacy 2: Natural language understanding with bloom embeddings. *Convolutional Neural Networks and Incremental Parsing*, 2017.

[16] E. Hoque, V. Setlur, M. Tory, and I. Dykeman. Applying pragmatics principles for interaction with visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):309–318, 2017.

[17] B. Jackson, T. Y. Lau, D. Schroeder, K. C. Toussaint, and D. F. Keefe. A lightweight tangible 3D interface for interactive visualization of thin fiber structures. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2802–2809, 2013.

[18] B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. In *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing*, pages 43–56, 1997.

[19] M. Kern, T. Hewson, A. Schätler, R. Westermann, and M. Rautenhaus. Interactive 3d visual analysis of atmospheric fronts. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1080–1090, 2018.

[20] B. Köhler, R. Gasteiger, U. Preim, H. Theisel, M. Gutberlet, and B. Preim. Semi-automatic vortex extraction in 4D PC-MRI cardiac blood flow data using line predicates. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2773–2782, 2013.

[21] A. Kumar, J. Aurisano, B. Di Eugenio, A. Johnson, A. Gonzalez, and J. Leigh. Towards a aialogue system that supports rich visualizations of data. In *Proceedings of Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 304–309, 2016.

[22] G. Li, M. Tian, Q. Xu, M. J. McGuffin, and X. Yuan. Gotree: A grammar of tree visualizations. In *Proceedings of CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[23] J. K. Li and K.-L. Ma. P4: Portable parallel processing pipelines for interactive information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(3):1548–1561, 2020.

[24] J. K. Li and K.-L. Ma. P5: Portable progressive parallel processing pipelines for interactive data analysis and visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1151–1160, 2020.

[25] J. K. Li and K.-L. Ma. P6: A declarative language for integrating machine learning in visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):380–389, 2021.

[26] R. Liu, M. Gao, S. Ye, and J. Zhang. IGScript: An interaction grammar for scientific data presentation. In *Proceedings of CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2021.

[27] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, 28(01):217–226, 2022.

[28] J. Ma, C. Wang, C.-K. Shene, and J. Jiang. A graph-based interface for visual analytics of 3D streamlines and pathlines. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1127–1140, 2014.

[29] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[30] G. A. Miller. WordNet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[31] A. Narechania, A. Srinivasan, and J. Stasko. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):369–379, 2020.

[32] A. Narechania, A. Srinivasan, and J. Stasko. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(02):369–379, 2021.

[33] B. Nsonga, M. Niemann, J. Fröhlich, J. Staib, S. Gumhold, and G. Scheuermann. Detection and visualization of splat and antisplat events in turbulent flows. *IEEE transactions on visualization and computer graphics*, 26(11):3147–3162, 2019.

[34] B. Nsonga, G. Scheuermann, S. Gumhold, J. Ventosa-Molina, D. Koschichow, and J. Fröhlich. Analysis of the near-wall flow in a turbine cascade by splat visualization. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):719–728, 2019.

[35] R. Peikert and F. Sadlo. Topologically relevant stream surfaces for flow visualization. In *Proceedings of Spring Conference on Computer Graphics*, pages 43–50, 2009.

[36] P. Rautek, S. Bruckner, M. Groller, and M. Hadwiger. ViSlang: A system for interpreted domain-specific languages for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2388–2396, 2014.

[37] C. Rössl and H. Theisel. Streamline embedding for 3D vector field exploration. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):407–420, 2012.

[38] F. Sadlo, R. Peikert, and E. Parkinson. Vorticity based flow analysis and visualization for Pelton turbine design optimization. In *Proceedings of IEEE Visualization Conference*, pages 179–186, 2004.

[39] T. Salzbrunn and G. Scheuermann. Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1601–1612, 2006.

[40] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2016.

[41] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, 2015.

[42] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang. Eviza: A natural language interface for visual analysis. In *Proceedings of Annual Symposium on User Interface Software and Technology*, pages 365–377, 2016.

[43] A. Srinivasan and J. Stasko. Natural language interfaces for data analysis with visualization: Considering what has and could be asked. In *Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Short Papers*, pages 55–59, 2017.

[44] A. Srinivasan and J. Stasko. Orko: Facilitating multimodal interaction for visual exploration and analysis of networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):511–521, 2017.

[45] Y. Sun, J. Leigh, A. Johnson, and S. Lee. Articulate: A semi-automated model for translating natural language queries into meaningful visualiza-

tions. In *Proceedings of International Symposium on Smart Graphics*, pages 184–195, 2010.

[46] J. Tao, C. Wang, N. V. Chawla, L. Shi, and S. H. Kim. Semantic flow graph: A framework for discovering object relationships in flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3200–3213, 2017.

[47] J. Tao, C. Wang, C.-K. Shene, and R. A. Shaw. A vocabulary approach to partial streamline matching and exploratory flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(5):1503–1516, 2016.

[48] H. Theisel, A. Friederici, and T. Günther. Objective flow measures based on few trajectories. *arXiv:2202.09566*, 2022.

[49] H. Theisel, T. Weinkauf, H.-C. Hege, and H.-P. Seidel. Saddle connectors - an approach to visualizing the topological skeleton of complex 3D vector fields. In *Proceedings of IEEE Visualization Conference*, pages 225–232, 2003.

[50] M. Tory and V. Setlur. Do what I mean, not what I say! design considerations for supporting intent and context in analytical conversation. In *IEEE Conference on Visual Analytics Science and Technology*, pages 93–103, 2019.

[51] Z. Wang, J. M. Esturo, H. Seidel, and T. Weinkauf. Stream line-based pattern search in flows. *Computer Graphics Forum*, 36(8):7–18, 2017.

[52] H. Wickham. ggplot2: Elegant graphics for data analysis. *New York, USA: Springer*, 2016.

[53] Q. Wu, T. Neuroth, O. Igouchkine, K. Aditya, J. H. Chen, and K.-L. Ma. A declarative grammar of flexible volume visualization pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1050–1059, 2018.

[54] Q. Wu, T. Neuroth, O. Igouchkine, K. Aditya, J. H. Chen, and K.-L. Ma. DIVA: A declarative and reactive language for in situ visualization. In *IEEE Symposium on Large Data Analysis and Visualization*, pages 1–11, 2020.

[55] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of annual meeting on Association for Computational Linguistics*, pages 133–138, 1994.

[56] L. Xu, T.-Y. Lee, and H.-W. Shen. An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1216–1224, 2010.

[57] L. Xu and H.-W. Shen. Flow web: A graph based user interface for 3D flow field exploration. In *Proceedings of IS&T SPIE Conference on Visualization and Data Analysis*, 2010.

[58] X. Ye, D. Kao, and A. Pang. Strategy for seeding 3D streamlines. In *Proceedings of IEEE Visualization Conference*, pages 471–478, 2005.

[59] B. Yu and C. T. Silva. FlowSense: A natural language interface for visual data exploration within a dataflow system. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1–11, 2019.

APPENDIX

1 ANALYSIS OF REJECTED QUERIES

We collected 26 rejected queries from a total number of 285 queries in our user study. The most common types of rejected queries (10/26) were the incomplete sentences, English grammar errors, and typos. The incomplete sentences were caused by pressing the "enter" key by mistake. Grammar errors do not necessarily lead to rejected queries. One observed rejected query was "show the flow of T in China with high", where the system failed to connect the attribute "T" with the constraint "high". The second most common type of rejected queries (6/26) was related to functionalities that were not implemented. For example, one participant wanted to visualize the volume, which was implemented after the user study. Six queries were rejected because of parsing failure. A typical example was using multiple adjectives consecutively, such as "cold fast north flow". The other four failure queries were due to mixing query formula with natural language, pressing the "enter" key repeatedly (two queries), and one unknown error. We added a visual hint to indicate that the query is being processed to avoid repeated enters after the study.

2 USER COMMENTS

The participants in our user study were positive with FlowNL in general. A typical comment was "the tool is inspiring and it can help me produce amazing animations in a short time". One participant mentioned that "I would like to deploy the tool on the super-computing platform I am working on. I believe it will attract a lot of users." For the participants with domain background, three users commented that they would like to use the tool in their research; while the other two stated that FlowNL might be useful for demonstration, but excessive in analyzing their data as they usually dealt with planar flows.

The participants also provided valuable suggestions in their comments to improve the tool. Some suggestions are already adopted in the latest version of FlowNL, such as visualizing objects in scalar fields, showing filter thresholds, enabling pan-and-zoom for 2D filters, and reducing the time for recycling particles of hidden objects, etc. Other suggestions are helpful but not fully incorporated yet. For example, removing derived objects from the interface, adding computational operations (e.g, computing average), and specifying global parameters, etc. Currently, we order the objects so that the hidden ones appear at the bottom. The other functions are not implemented yet.

3 OBJECT AND ATTRIBUTE SPECIFICATION

Users may incorporate various kinds of objects by specifying the primitive objects and their attributes in a meta file. Currently, FlowNL supports unstructured point clouds and structured grids in multiple file formats, such as VTK (visualization tool kit), NC (network common data form), raw binary, and raw textural files. Figure 1 shows an example of primitive object definition for the ECMWF data set. This data set contains four types of primitives: flow patterns, grid (flow and scalar fields), geographic regions, and streamlines. The streamlines and their attributes are automatically generated by the system. Therefore, the streamline primitive does not appear in the meta file.

A primitive contains several fields, including type, name, attributes, derives, file paths, file type, etc. The most important one is the attributes field, as this field defines the "content" of a primitive. FlowNL provides multiple ways to flexibly specify the data of an attribute. We highlight four different types of attribute specification in Figure 1. Figure 1 (a) shows the attribute "latent (vector)" in the flow pattern primitive. In addition to the name "latent", this specification also contains a type value "FLOAT128", which indicates that the attribute is a 128-dimensional vector of floating-point numbers. The "latent" and "radius" attributes of an element jointly defines a high-dimensional sphere in the deep representation space. The deep representation is produced by an autoencoder that encodes the normalized distance matrices of streamline points. The normalized distance matrix uniquely represents the shape of a streamline segment, which is invariant under rigid transformations. Therefore, the latent vector can be used to describe the shape of a streamline segment. To represent irregular regions, multiple spheres

```
{
    "type": "unstructured",
    "name": "flow pattern",
    "attributes": [{"name": "latent", "type": "FLOAT128"},   a
        {"name": "radius", "type":"FLOAT"},
        {"name": "ID", "type":"INT"}],
    "derives": [{"name": "spiral"},{"name": "circulation"},{"name": "turbulence"}],
    "file path": "patterns.dat",
    "file type": "raw"
},
{
    "type": "structured",
    "name": "grid",
    "attributes": [{"name": "pv"},{"name": "q"},{"name": "vo"},{"name": "d"},{"name": "r"}   b
        {"name": "t"},{"name": "clwc"},{"name": "ciwc"},{"name": "cc"},{"name": "o3"},
    c {"name": "geo position", "variables":[{"name":"longitude"}, {"name":"latitude"}]},
        {"name": "flow", "variables":[{"name":"u"}, {"name":"v"}, {"name":"w"}]},
    d {"name": "vel. mag.", "definition":{"operator":"magnitude", "attribute":"flow"}}],
    "dimension names":{"x": "longitude", "y": "latitude", "z": "level", "t": "time"},
    "dimension filters":{"t": "16"},
    "file path": "201809.nc",
    "file type": "nc"
},
{
    "type": "unstructured",
    "name": "geographic region",
    "attributes": [{"name": "geo position", "type": "FLOAT2"},
        {"name": "radius", "type": "FLOAT"},
        {"name": "ID", "type": "INT"}],
    "derives": [{"name": "china"},{"name": "indian ocean"},{"name": "mediterranean sea"}],
    "file path": "georegions.txt",
    "file type": "raw-text"
}
```

Fig. 1. The primitive object specification for the ECMWF data set.

are used. A preserved attribute "ID" is used to correspond the spheres to the derives.

Figure 1 (b) shows a simpler example of defining an attribute "r" of the primitive "grid". This primitive contains the flow and scalar fields in an NC file. As the NC format is self-described, it does not specify the dimension or data type of the attribute. Our data engine can reorganize the variables in the NC file to form attributes as well. Figure 1 (c) illustrates such an example. The attribute "geo position" (geographic position) is defined by concatenating two variables "longitude" and "latitude" into 2D vectors. The data engine will span the low dimensional attributes to the entire grid. The data engine can also derive attributes from existing ones, as shown in Figure 1 (d). In this example, the attribute "vel. mag." (velocity magnitude) is derived as the magnitude of the "flow" (the vector field).

The definition of primitive "geographic region" is similar to the flow pattern. The only difference is that geographic regions are 2D circles. Users may use similar specifications to define detected features as point clouds. Note that we do not support unstructured mesh data in our current implementation. As the objects are mainly used to specify regions, the connections among points are less relevant in the current version of FlowNL. We plan to incorporate mesh data in the future extension, so that the integral surfaces in flow fields and isosurfaces in scalar fields can be visualized.