# An Overview of the Natural Language Interaction Protocol

Sanjay Aiyagari[1], Rithik Babu[2], Elisa Bertino[3], Jan Bieniek[4], Yan-Ming Chiou[5], Raj Dodhiawala[6], Erik Erlandson[1], Sugih Jamin[2], Ashish Kundu[7], Jon Lenchner[8], Tejas Maire[2], Matthew L. Mauriello[9], Mohamed Rahouti[4], Abhay Ratnaparkhi[8], Tom Sheffler[6], Chien-Chung Shen[9], Dinesh Verma[8], Wenpeng Yin[10], Luyi Xing[11], Jinjun Xiong[12], Hasan B. Zengin[2]

[1]Red Hat, [2]University of Michigan, [3]Purdue University, [4]Fordham University, [5]SRI International, [6]Independent, [7]Cisco Systems, [8]IBM, [9]University of Delaware, [10]Pennsylvania State University, [11]Indiana University, [12]University at Buffalo.

saiyagar@redhat.com, rithikb@umich.edu, bertino@purdue.edu, jbieniek@fordham.edu, yan-ming.chiou@sri.com, raj.dodhiawala.work@gmail.com, eerlands@redhat.com, sugih@umich.edu, ashkundu@cisco.com, lenchner@us.ibm.com, tmaire@umich.edu, mlm@udel.edu, mrahouti@fordham.edu, abhay.Ratnaparkhi1@ibm.com, tom.sheffler@gmail.com, cshen@udel.edu, dverma@us.ibm.com, wenpeng@psu.edu, luyixing@iu.edu, jinjun@buffalo.edu, hbzengin@umich.edu

**Abstract**

The ability of generative AI to ingest a large corpus of data and use that to perform tasks such as translation, sentence transformation, and describe new content can give a tremendous boost to the economic and social benefits to society. To unlock the true potential of this technology, a commonly used protocol that allows client machines with AI agents to talk to server machines running their own AI agents, to create new data and describe content, is critical. This protocol, which needs to be both a de-jure and a de-facto standard, must be defined and designed in an open community with participation from interested technical parties and similar stakeholders. A group of researchers from various companies and universities has come together to define such a standard application-level protocol–the Natural Language Interaction Protocol (NLIP), and implement open-source implementations of the same. In this paper, we provide the motivation for NLIP, its requirements, and outline its initial specification.

**Code** — https://github.com/nlip-project
**Extended version** — https://github.com/nlip-project/documents/blob/main/NLIP_Specification.pdf

## Introduction

The technology of Generative AI (GAI) (Bandi 2023) has the potential to be truly transformative to society. Despite some limitations such as "hallucinations," the technology is capable of many functions, including but not limited to answering questions, translating, describing and summarizing content, and generating new content. This enables the creation of intelligent agents (Alonso 2002) that can use AI to analyze data and provide new services.

A much bigger boost to the social benefits of GAI technology can be obtained by interaction among different intelligent agents, which may be under the control of different organizations and users. The interaction among intelligent agents can unlock new economic and social value, just like the interactions among various Internet-based services was enabled with the advent of the web browser.

For the intelligent agents to interact with each other, we need a standard common protocol that is used widely among interacting agents. The potential of the Internet was unlocked by the creation and adoption of the Hypertext Transfer Protocol (HTTP) (Berners-Lee 1996). To unlock the potential of GAI, we need an equivalent ubiquitous protocol that intelligent agents can use to communicate with each other.

To address this need, researchers from multiple organizations have come together to define and implement this protocol. The protocol is called the Natural Language Interaction Protocol (NLIP). In this paper, we provide an overview of the requirements for the design of NLIP, the different ways NLIP can be deployed among intelligent agents, and an overview of its design.

The wide adoption of such a protocol can deliver many benefits to society. Just like a plethora of client-side applications were consolidated into a single web browser application during the advent of the Internet, NLIP can reduce the explosion of mobile applications that make current hand-held systems confusing and complex to use. It can enable a common mechanism for client applications to access various business applications. It can simplify the effort required to provide GAI enabled business services, and it can enable the consumers of the application to use GAI to manage the various services they access.

We are defining this protocol and implementing proof of concept endpoints supporting this protocol as an open-source collaborative project. The specifications and code, both of which are evolving are maintained at https://github.com/nlip-project.

The rest of this paper deals with the requirements, deployment models, and initial specification of the NLIP protocol.

## Requirements

To be adopted and used widely, NLIP needs to satisfy several requirements. In this section, we enumerate some of these requirements, which have been categorized into the three broad areas of adaptability, security, and performance.

### Adaptability Requirements

The adaptability requirements cover the requirements needed to support the protocol across many different platforms and implementation choices that may be made.

*Multi-Platform Support:* Many different platforms are used by various organizations. Among the hand-held devices (Okediran 2014), iOS and Android are two (but not the only) common platforms used across many devices. On laptops, we have systems based on MacOS, Linux, and Windows as some of the widely deployed platforms. On the server side, Linux is the dominant platform, but there are several others, such as Microsoft Windows IIS and z/OS on mainframes, which are also important to consider. We need to design NLIP so that it does not depend upon the characteristics of a specific operating system (Silberschatz 1991).

*Multi-Language Support*: Software systems are written in many different programming languages (Kumar and Dahiya 2017), which include but are not limited to Java, Javascript, Python, Go, C/C++, C#, Kotlin and Swift. We need to design the protocol to be implemented efficiently in any programming language. Similarly, the users of the computer come from a variety of backgrounds and may prefer to use their native natural language for communication. NLIP should be designed so that it can work across many different natural languages.

*Multi-Transport Support*: Distributed applications may communicate over many different network protocols. Common choices used in application development include REST over HTTPS, WebSockets, and QUIC. However, other protocols may emerge over time, and NLIP should be designed to work over any underlying transport protocol.

At the same time, NLIP design should not reinvent the wheel. Many of the commonly prevalent protocols on the Internet provide excellent solutions for performance and security considerations, and NLIP design should leverage and build upon the capabilities of the underlying protocol.

*Multi-Modal Content Support*: NLIP should support content transfer in many different modalities. Text or natural language provides a common content modality that needs to be supported, but there are other modalities that are needed. Many business services may require image, audio or video content, while others may need specialized content such as location coordinates, sensor readings, and other field-collected data. NLIP should support all modalities of the content that may be needed.

### Security and Privacy Requirements

Security and Privacy (Li, Bertino and Yi 2014) requirements deal with the critical task of allowing Internet-based systems to provide their services to good actors efficiently while preventing bad actors from harming, disrupting, or blocking access to the system or other users. NLIP is designed to support the following security requirements:

*Anonymous Mode*: NLIP should allow an agent to interact with another agent in an anonymous mode. The intention to interact in an anonymous mode must be agreed upon by all agents in the interaction, even if a party does not require anonymity itself.

*Authentication and Authorization Support*: While supporting anonymous mode of interactions, NLIP should also enable authentication and authorization among agents who require them. There are multiple viable authentication and authorization mechanisms in use throughout the Internet. NLIP will leverage existing mechanisms and enable seamless access to them.

*Encryption of data in motion*: NLIP should support communication over an encrypted channel. As with existing services, NLIP is designed to leverage existing secure protocols such as TLS when applicable.

*Prevention of Denial of Service*: AI-based services, especially those that rely on large language models are susceptible to various denial-of-service attacks (Mahjabi et. al. 2017). These include but are not limited to malicious users leaving large amounts of unnecessary context stored at servers or trying to overwhelm the server with multitudes of requests. Mechanisms such as rate control or limits on context storage must be adopted to prevent such attacks. The protocol must enable the enforcement of these constraints.

*Regulatory Compliance*: Many services may need to provide information about the policies they are operating under. NLIP must enable the intelligent agents to easily define and exchange their privacy, data retention, or other policies they may be using

### Performance Requirements

Performance requirements ensure that the protocol enables efficient usage of resources and minimizes user-perceived latency. To enable good performance, NLIP must satisfy the following requirements:

*Context Management:* The context of an interaction among agents is the prior history of interaction among those agents, influencing how an intelligent agent may respond to a request. NLIP must support capabilities for dynamic context management. These may include negotiating how much context history may be supported in a session or switching the responsibility of who stores the context among the different communicating agents.

*Streaming Support:* To maintain good performance, the protocol must enable streaming mode of communication in an asynchronous mode as well as a synchronous mode of communication.

*Control and Data Separation:* Some of the exchanges among intelligent agents may deal with the issue of controlling the communication (e.g., examining the policies offered by the other agent or negotiating context management). In contrast, others may deal with the actual function the two agents want to carry out. We refer to the former as control and the latter as data. Agents may choose to handle the two types of exchanges separately, and NLIP must provide a clear demarcation between the two.

## Deployment Models

A deployment model refers to the configuration of various agents in which NLIP may be used. We envision NLIP supporting a range of deployment models, some of which are outlined in this section.

### Client Server Model

The client-server deployment model is the traditional interaction model among two agents in which one agent acts as a client of the other agent. The server agent waits for the client agent to initiate communication. This traditional model is shown in Figure 1.

One, both, or none of these agents may be leveraging a LLM for their operation. If they are leveraging a LLM, the agent may or may not want to expose details about the LLM they are using to the other agent. The typical model for an end-point in the server (or the client configuration) would be not exposing the LLM. In those cases, the configuration would look like the NLIP-Proxy configuration described in the next section.

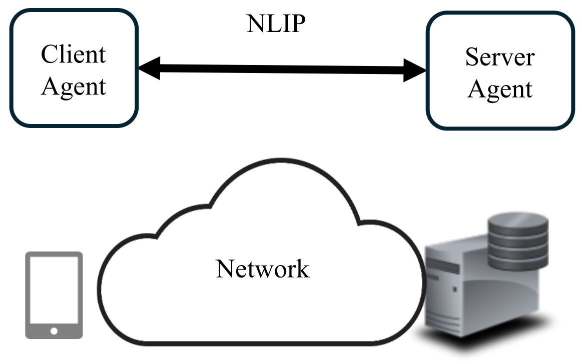

Figure 1. The Client-Server model for NLIP

### NLIP Proxy Configuration

In the NLIP Proxy Configuration, one of the agents uses NLIP to enable a proxy interface for another service. The service that is used may be an existing application that uses its own proprietary protocol for communication.

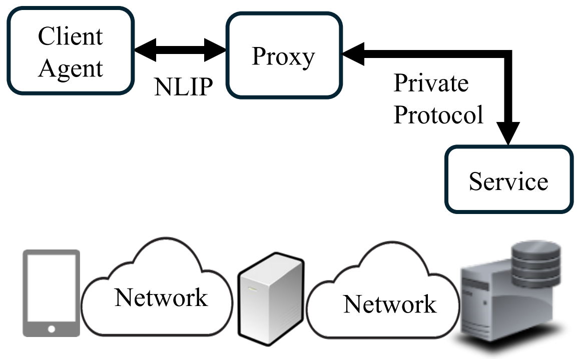

Figure 2. The Proxy configuration for NLIP.

The NLIP Proxy configuration may be used to provide a common interface to existing LLM-based conversation services. Currently, each conversational service on the Internet uses its own proprietary API to communicate with its clients. Replacing those APIs with a standard NLIP interface can enable a single client application to interact with several conversational services. A conversational service may opt to adopt NLIP directly, leading to the configuration in Figure 1, or may choose to implement the proxy configuration to achieve the same result.

### Federator Configuration

In the federator configuration, the NLIP Proxy enables a common interface to many different backend services. As an example, a common NLIP conversational agent may provide an interface to access many common conversation services available on the Internet.

The federator configuration may also provide its own value-added services by combining the responses from many different backend services into a single response to the user. Some examples of such services in the conversational context may include using one of the backend services to provide guard-rails against hallucination by the other service.

The federator may also provide a way to integrate many existing Internet services into a single intelligent service. For example, a federator may search across several web-based merchants to find the best price for a user interested in shopping for a particular merchandise item.

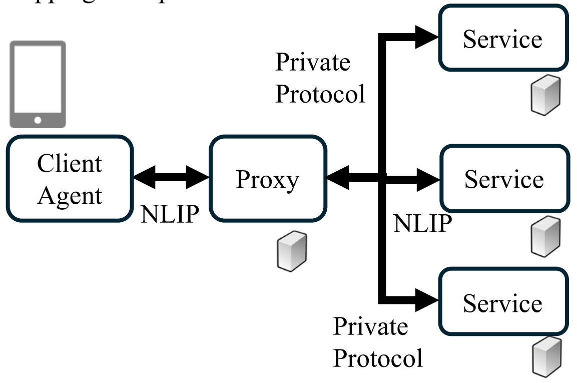

Figure 3. A NLIP Server in a federator configuration, where it is a front-end integrating many existing services.

Note that the federator pattern may use NLIP between the proxy and some of the back end services, and use proprietary protocol with other back-end services.

### Back-Level Configuration

One or more of the agents may not have sufficient computing resources to run AI models. This may be true for some agents running on a mobile phone, a hand-held device, or embedded Internet-of-Things systems which may not have the ability to run a large language model. In those cases, the client agent may choose to leverage a local service with the ability to use the LLM. Alternatively, the client agent may request the server agent to use a limited vocabulary which can be interpreted easily by the agent. This configuration is shown in Figure 4.

In the back-level configuration, the local LLM Service shown may be another service running on the Internet and need not be co-located with the client machine.

One common use of the back-level configuration may be for a weak hand-held device to provide voice modality and convert that to text. Instead of using voice modality within the NLIP communication with the server agent (which is a modality supported by NLIP), the client may choose to use a local LLM service to convert speech to text and interact with the service agent using text modality.

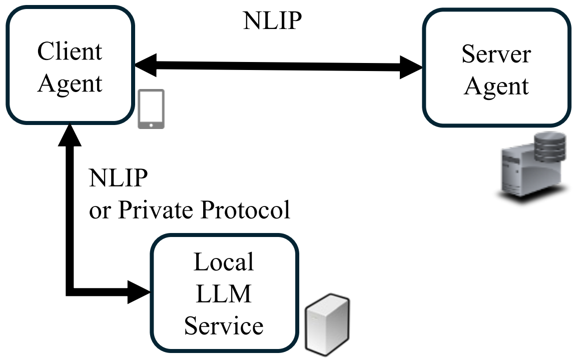

Figure 4. Back-level Configuration of NLIP, where a client uses a local LLM service for its functions.

## A Future Scenario

We want to discuss NLIP's impact on today's computing landscape by providing a scenario of how things are done today, and how they might be done in the future.

### The Current State

We consider the situation of an academic traveling to various cities to attend scientific conferences of interest.

At present, almost every major city has a public transportation system that publishes its own mobile application for the benefit of its riders. The application provides many capabilities, including an easy view of  train timetables and status, buying ride tickets/passes, finding how to go from point A to point B. Since every city provides its own application, the academic needs to find and install this application for every city being visited, which is relevant only for the period of stay in the city. If the academic is not a frequent visitor to the cities, the need to install the plethora of apps is a significant burden.

The academic is also encouraged by each conference to download and install a conference application to make the participation experience better. The lifetime of the conference application is that of a few days, and the experience and interfaces of each application are very different. The academic does not wish to install various applications but is left with little choice.

The same is true of the applications being requested by the hotels where the academic is staying, or the airlines that the academic may be using for travel expenses.

Not only are these applications superfluous and a nuisance for the academic, they also impose a significant burden on the businesses that need to provide them. If the cities, conferences, hotels and airlines can obtain all of their business value with a single common application without needing to develop their own, their IT expenses and support requirements would decrease in a non-trivial manner.

### The Future State

We envision a future with NLIP where the academic has a single application which is enabled to communicate using NLIP. The conference organizers, the hotel chains and the city taxi operators do not need to provide their own private applications. Instead, they provide their own NLIP server using either a NLIP client-server configuration model or a NLIP Proxy Server model.

The application that the academic uses enables both text based and multimodal content exchange with the server. This allows the academic to manage all of the travel arrangements using a single common application. There is no reduction in the functionality available, just the added convenience of a single application for the academic. For the businesses that are providing services to the academic, their costs for supporting the IT required for operation is reduced significantly.

## The Current NLIP Specification

In this section we describe the current specification for NLIP. Over time, the NLIP specification will evolve but even the current draft can support a large number of inter-agent communications.

NLIP follows a request-response paradigm, as opposed to a remote-procedure call paradigm, in which clients send requests to servers and receive a response back. In NLIP, the client is the entity that initiates the communication, and the server is the entity that waits for requests from one or more clients. The server must be willing to receive communication requests from any client.

Requests and responses are generally exchanged using the JSON format. The JSON format is excellent for carrying text information but can become inefficient and cumbersome when carrying large binary data, or structured content such as XML or HTML, which require masking of special symbols such as quotations. Therefore, NLIP permits the transfer of such information using underlying protocols such as HTTP.

NLIP supports enforcement of authentication and authorization information among clients and servers. There are many underlying mechanisms that can be used for authentication and authorization. Some servers may choose to communicate with anyone without authentication or authorization, while others may enforce that each message be authenticated. Tokens for authentication and authorization are supported by NLIP, but they are considered opaque base64-encoded text strings which are used and interpreted by underlying security mechanisms.

### NLIP JSON Messages

The majority of exchanges between client and servers happen using a JSON message with the following fields:

- control: An optional boolean value to indicate whether the message is a control or data message. Example control messages could be a query of server policy, to negotiate parameter configurations, etc. In an end-point with a human user, the content of data messages is normally relayed to the human user, whereas control messages would normally be handled by the end point software in a manner transparent to the human user. When this optional value is missing, the end-point needs to infer this value from the content of the messages.
- format: This required field specifies the format of the content. It has to take one of the values specified in the "Allowed values of the 'format' field" subsection below.
- subformat: This required field specifies a further refinement of the format field. It can take a value that makes sense for the type of format as described in the ensuing "Allowed values of the 'format' field" subsection.
- content: This required field includes the actual content that is being sent between the client and the server.
- submessages: an optional field whose value is a JSON array containing one or more valid NLIP sub-messages. A sub-message contains only the format, subformat and content field as described above.

We anticipate that the bulk of messages will not include any submessages. Submessages may be used when sending multi-modal content such as a request from a client to deposit a check to a banking service, with images of the check attached.

### Allowed values of the "format" field

The following are the allowed values for the format fields in an NLIP JSON message.

- text: The format field of 'text' indicates that the content is natural language text in some language. The subformat specifies the natural language used, e.g., 'english'. Capitalization is not important in the subformat.
- token: The format field of 'token' can be used to carry opaque tokens to serve a variety of purposes including session identification, authentication verification, authorization enumeration, or any other operations to enable a natural-language interaction session. The subformat field indicates the type of the token. Subformats starting with

the prefix of ‘authentication’ or ‘conversation’ are to be used for the purpose of carrying authentication tokens and conversation identifiers, with latter part of the subformat string containing any additional data an end-point may want to introduce. The subformat can also be any string which the end-point creating the token uses for its convenience. The content field of a token submessage is also opaque to NLIP. A NLIP message may carry zero, one, or more submessages with the ‘token’ format. An end-point receiving a token submessage must include the identical token submessage in the next message sent to the peer end-point.

- structured: The format field of ’structured’ indicates that the content contains structured information, i.e., the subformat is one of ‘json’, ‘uri’, ‘xml’, ‘html’. The content is a URI if the subformat is ‘uri’, or an encoded string which contains an embedded content in the specified subformat. The uri subformat can be used to support many protocols such as ’http’, ’https’, ‘websockets’, ‘WebRTC’, etc.
- binary: the subformat could be one of ‘audio/<encoding>’, ‘image/<encoding>’, ‘sensor/<encoding>’, or ‘generic/<encoding>’, where the ‘<encoding>‘ is the original encoding of the binary data, e.g., bmp, gif, jpeg, jpg, png, tiff, etc. for images, mp3 for audio, or any other binary encoding applicable to the type of the binary data recognized by both client and server. The content field carries the binary data that has been base64 encoded. The binary format is intended for small binary data: for example, a few seconds of audio clips, or small icons or thumbnails, that can be efficiently transferred as base64-encoded text. Upload of large binary data is addressed in the next subsection.
- location: the subformat can be one of ‘text’ or ‘gps’. If the subformat is ‘text’, a textual description of the location, e.g., “221B Baker St., London, UK”, must be included in the content field. If the subformat is ‘gps’, GPS coordinates must be included in the content field.
- generic: the subformat and content can be any generic entries which the client and server mutually understand. The generic format provides message format extensibility to NLIP.

In all of the above keywords, capitalization is not important. Both the client and server must accept the keywords regardless of the mixture of capitalization in the fields of format and subformat.

## Upload of Large Binary Files

While small binary data can be transferred as base64-encoded data in a JSON message, it may be more efficient to transfer large binary data directly from capture device or storage to the network, not as JSON messages. Similarly, encoding of large HTML or XML files into a JSON string may require significant complexity.

When a NLIP server needs to send a large amount of data to or from the client, it provides an NLIP message with format: structured and subformat: uri to tell the client which URI endpoint(s) to retrieve or to upload the large data. The large data may be binary, HTML, XML or another format and encoding.

When a client (i.e., an end-point that cannot export a URI to download the content) needs to send a large binary file, it can ask the server for an URI to upload the large content. In those cases, the server can provide a URI for the large content to be uploaded. For example, in the HTTP(S) case, the large data can be sent using HTTP Content-Type: multipart/form-data to a URI that expects multipart/form-data.

## NLIP Binding to REST Interfaces

NLIP may be bound to a variety of communication protocols. This section provides an exemplar binding to a REST API running on top of HTTPS.

The following considerations regarding HTTPS and REST API handling inform our initial design of NLIP:

1. HTTP(S) server access cannot be routed/demultiplexed based on the content of the incoming message due to use of (de)serialization libraries; if routing is needed, it must be by URI endpoint (and/or its query components) so that message routing can be done before message parsing, instead of requiring multi-pass parsing.
2. HTTP(S) clients cannot route messages by URI. If NLIP has different versions, any server’s response must be of the same version the client used in its request. Consequently, information that can change from one response message to the next, such as the value of NLIP’s control field, cannot be specified as a part of the URI, nor its query component, but must be specified in the message itself.

Each HTTPS server implementing NLIP, for example a server with address example.com and port 5550, must export a fixed, well-known nlip end point, in this case, https://example.com:5550/nlip. On this primary end-point, the NLIP server must accept a client request which contains a NLIP message with the ’format’ field of ’text’, and respond to it. The response must either indicate that the server is refusing the connection, or the result of processing the request. The NLIP server may direct the client to additional end points for upload of client data.

## Current Status

The NLIP protocol is currently being actively worked on by the authors of this article. An initial implementation of the protocol with some exemplar clients and server implementations are available publicly in github. The development of the protocol and endpoint implementations are being done collaboratively in open source so that the protocol is available publicly to all interested parties.

There are plans to take the NLIP activity to an official standard body so that it can be defined as an official standard protocol. The initial plan is to pursue standardization as a technical committee of European Computer Manufacturers Association (ECMA).

## References

(Bandi 2023) Bandi, A., Adapa, P.V.S.R. and Kuchi, Y.E.V.P.K., 2023. The power of generative AI: A review of requirements, models, input–output formats, evaluation metrics, and challenges. *Future Internet*, *15*(8), p.260.

(Berners-Lee 1996) Berners-Lee, T., Fielding, R. and Frystyk, H., 1996. RFC1945: Hypertext Transfer Protocol--HTTP/1.0

(Alonso 2002) Alonso, E., 2002. AI and Agents: State of the Art. AI Magazine, 23(3), pp.25-25

(Okediran 2014) Okediran, O.O., Arulogun, O.T., Ganiyu, R.A. and Oyeleye, C.A., 2014. Mobile operating systems and application development platforms: A survey. International journal of advanced networking and applications, 6(1), p.2195.

(Silberschatz 1991) Silberschatz, A., Peterson, J.L. and Galvin, P.B., 1991. Operating system concepts. Addison-Wesley Longman Publishing Co., Inc.

(Kumar and Dahiya 2017) Kumar, K. and Dahiya, S., 2017. Programming languages: A survey. International Journal on Recent and Innovation Trends in Computing and Communication, 5(5), pp.307-313.

(Li, Bertino and Yi 2014) Li, X., Bertino, E. and Yi, M., 2014. Security of new generation computing systems. Concurr. Comput., 26(8), pp.1475-1476.

(Mahjabi et. al. 2017) Mahjabin, T., Xiao, Y., Sun, G. and Jiang, W., 2017. A survey of distributed denial-of-service attack, prevention, and mitigation techniques. International Journal of Distributed Sensor Networks, 13(12).