Individual Submission
Internet-Draft
Intended status: Informational
Expires: 10 August 2026

D. Levy
DanLevy Consulting

6 February 2026

The "llm" URI Scheme
draft-levy-llm-uri-scheme-00

Abstract

This document defines the "llm" Uniform Resource Identifier (URI) scheme for identifying Large Language Model (LLM) endpoints and their configuration. The scheme encodes provider host, model identifier, authentication credentials, and inference parameters into a single, portable connection string, analogous to database connection URIs such as those used by PostgreSQL and MongoDB.

This document registers the scheme name in the IANA "Uniform Resource Identifier (URI) Schemes" registry as a provisional registration per RFC 7595.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 10 August 2026.

Copyright Notice

Table of Contents

1. Introduction

Large Language Model (LLM) APIs have become a foundational component of modern software systems. As of 2025, applications commonly integrate with multiple LLM providers including OpenAI, Anthropic, Google, Mistral, Cohere, and self-hosted models via frameworks such as Ollama and vLLM.

Currently, there is no standardized way to represent an LLM endpoint configuration as a single, portable identifier. Developers typically manage provider connections through a proliferation of environment variables (e.g., OPENAI_API_KEY, ANTHROPIC_API_KEY, MODEL_NAME, API_BASE_URL, TEMPERATURE, MAX_TOKENS) that vary by provider, framework, and deployment context.

This document defines the "llm" URI scheme to address this fragmentation by encoding all connection parameters into a single URI string, following the well-established precedent of database connection URIs.

1.1. Motivation

The database ecosystem underwent a similar standardization journey. Early database clients required separate configuration for host, port, username, password, database name, and various driver options. The adoption of connection URIs (e.g., postgres://user:pass@host:5432/dbname) dramatically simplified configuration, improved portability, and enabled tooling standardization.

The LLM ecosystem is at an analogous inflection point. A single connection string encoding provider, model, authentication, and inference parameters would provide:

* *Portability:* A single string that moves between environments, scripts, and deployment targets.

* *CLI ergonomics:* One argument instead of many flags.

* *Language agnosticism:* Every programming language includes a robust URI parser per RFC 3986, providing parsing, validation, and sanitization without additional dependencies.

* *Framework interoperability:* A common configuration format that middleware, routers, and observability tools can consume.

1.2. Relationship to Other Work

The "ai" URI scheme (draft-sogomonian-ai-uri-scheme) defines a general-purpose scheme for identifying AI-addressable resources such as agents, tools, and tasks, with its own resolution protocol (AIIP). The "llm" scheme defined in this document is narrower in scope: it is a configuration-oriented URI for identifying a specific model endpoint and its inference parameters, intended to be resolved by client libraries into standard HTTPS API calls. The two schemes are complementary and non-overlapping.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

LLM: Large Language Model. A neural network trained on large text corpora that generates text completions given an input prompt.

Provider: An organization or service that hosts and serves an LLM via an HTTP API (e.g., OpenAI, Anthropic, a self-hosted Ollama instance).

Model: A specific LLM offered by a provider, identified by a model name or version string (e.g., "gpt-5", "claude-sonnet-4", "llama3").

Inference Parameters: Configuration values that control the model’s output behavior for a given request, such as temperature, top-p, and maximum token count.

Connection String: A URI conforming to this specification that encodes a complete LLM endpoint configuration.

3. URI Syntax

3.1. The "llm" Scheme

The "llm" URI scheme uses the generic URI syntax defined in [RFC3986] Section 3, with all standard components:

```
llm-URI = "llm:" "//" authority path-abempty
          [ "?" query ] [ "#" fragment ]
```

The double-slash "//" is used because the scheme defines a hierarchical structure with a naming authority (the provider host).

Example:

llm://api.openai.com/gpt-5?temp=0.7&max_tokens=1500

This identifies a GPT-5 model endpoint at api.openai.com with temperature 0.7 and a maximum output length of 1500 tokens.

3.2. ABNF

Using the ABNF notation of [RFC5234] and the URI components defined in [RFC3986]:

```
llm-URI       = "llm:" "//" llm-authority
                llm-path [ "?" query ] [ "#" fragment ]

llm-authority = [ userinfo "@" ] host [ ":" port ]

llm-path      = "/" model-id *( "/" sub-path )

model-id      = 1*( unreserved / pct-encoded / "." / "-" )

sub-path      = 1*( unreserved / pct-encoded / "." / "-" )
```

Where "userinfo", "host", "port", "query", "fragment", "unreserved", and "pct-encoded" are as defined in [RFC3986].

4. URI Components

4.1. Scheme

The scheme component MUST be "llm" (case insensitive per [RFC3986] Section 3.1, but MUST be registered and SHOULD be written in lowercase).

4.2. Authority

The authority component identifies the LLM provider and follows the syntax defined in Section 3.2 of [RFC3986]:

```
authority = [ userinfo "@" ] host [ ":" port ]
```

4.2.1. Userinfo

The userinfo subcomponent, if present, provides authentication credentials for the provider API:

```
userinfo = app-label [ ":" api-key ]

app-label = 1*( unreserved / pct-encoded )
api-key   = 1*( unreserved / pct-encoded )
```

The "app-label" field identifies the application, project, or organization associated with the credential. Client libraries SHOULD map this value to the provider's organization or project header where applicable. For example, OpenAI-compatible APIs accept an OpenAI-Organization header; the app-label SHOULD be sent as the value of this header (or the provider's equivalent, such as Anthropic-Organization or X-Project-ID). This enables providers to attribute usage, enforce per-project rate limits, and route billing.

The "api-key" field contains the API secret or token.

Example:

```
llm://my-app:sk-proj-abc123@api.openai.com/gpt-5
```

Implementations that produce or log LLM URIs MUST sanitize the userinfo component to prevent credential leakage (see Section 10).

4.2.2. Host

The host identifies the provider's API endpoint.

4.2.3. Port

The port subcomponent is OPTIONAL. When omitted, clients SHOULD use port 443 (HTTPS) for remote providers and port 11434 for localhost (the conventional Ollama default). Implementations MAY define other defaults.

4.3. Path (Model Identifier)

The path component identifies the model. The first path segment MUST be the model identifier. Additional path segments MAY be used for provider-specific namespacing (e.g., organization or deployment identifiers).

The model identifier SHOULD correspond to the model name string accepted by the provider's API.

Examples:

```
/gpt-5                          -- Simple model name
/gpt-5.2                        -- Versioned model name
/anthropic/claude-sonnet-4.5    -- Namespaced model (e.g., for
                                   aggregator providers like
                                   Bedrock or Vercel AI)
/llama3                         -- Self-hosted model
```

Clients MUST NOT interpret path segments beyond mapping them to the provider's model selection parameter. Path semantics are defined by the provider.

4.4. Query Parameters

The query component carries inference parameters and provider-specific options as key=value pairs separated by "&", following standard URI query syntax.

This document defines a set of well-known parameter names (Section 6) that implementations SHOULD recognize. Providers and clients MAY define additional parameters. Unrecognized parameters SHOULD be passed through to the provider API where possible and MUST NOT cause a client error.

Parameter names are case-sensitive. Values MUST be percent-encoded per [RFC3986] Section 2.1 when they contain reserved characters.

Example:

```
?temp=0.7&max_tokens=1500&top_p=0.9&cache=true
```

For simple structured parameters, implementations SHOULD use dot-notation to flatten nested keys:

```
?web_search.max_uses=3&web_search.enabled=true
```

This avoids the brittleness of encoding full JSON objects in query strings. When dot-notation is insufficient (e.g., deeply nested or array-valued configurations), implementations MAY percent-encode a JSON value:

```
?web_search=%7B%22maxUses%22%3A3%7D
```

Which represents: web_search={"maxUses":3}

Implementations that accept JSON-valued query parameters SHOULD impose a maximum length on individual parameter values and MUST be aware that many HTTP intermediaries (proxies, CDNs, load balancers) enforce URI length limits, commonly between 2,048 and 8,192 octets.

Complex configurations that would exceed practical URI length limits SHOULD be provided via out-of-band configuration files or environment variables rather than encoded in the URI.

4.5. Fragment Identifier

The fragment component, if present, is client-side and is NOT sent to the provider. Fragments MAY be used by client applications for purposes such as identifying a specific system prompt variant, a named configuration profile, or a UI context.

Example:

llm://api.openai.com/gpt-5?temp=0.7#coding-assistant

The semantics of the fragment are not defined by this specification and are left to the consuming application.

5. Semantics and Operations

5.1. Resolution

An "llm" URI does not resolve to a resource in the traditional HTTP sense. Instead, it is a configuration identifier that a client library parses to construct API requests to the identified provider.

The resolution process is:

1. Parse the URI per [RFC3986].

2. Extract the host to determine the provider API base URL. The transport protocol MUST be HTTPS for remote hosts. Clients MAY use HTTP for localhost/loopback addresses only.

3. Extract the path to determine the model identifier.

4. Extract userinfo, if present, to obtain authentication credentials.

5. Extract query parameters and map them to the provider’s API request format.

6. Construct an HTTPS request to the provider’s inference endpoint (e.g., POST /v1/chat/completions for OpenAI-compatible APIs).

Clients MUST NOT dereference an "llm" URI by making a GET request to the URI itself. The URI is a configuration descriptor, not a resource locator.

5.2. Operations

The primary operation associated with an LLM URI is inference: sending a prompt to the identified model and receiving a completion. This document does not define the prompt or response format, which varies by provider and is outside the scope of URI scheme definition.

Implementations MAY support additional operations such as:

* Model metadata retrieval (listing capabilities, context window size, supported parameters).

* Streaming inference (server-sent events or chunked transfer).

* Batch inference.

* Embedding generation (when the identified model supports it).

6. Well-Known Query Parameters

The following query parameter names are RECOMMENDED for interoperability. Implementations SHOULD map these to the corresponding provider API parameters.

Parameter	Type	Description
temp	float	Sampling temperature (0.0 to 2.0)
max_tokens	int	Maximum tokens in the completion
top_p	float	Nucleus sampling threshold (0.0 to 1.0)
top_k	int	Top-k sampling parameter
stop	string	Stop sequence(s), comma-separated
seed	int	Random seed for reproducibility
stream	bool	Enable streaming response
timeout	int	Request timeout in milliseconds
retries	int	Maximum retry attempts
cache	bool	Enable response caching
format	string	Response format (e.g., "json")
system	string	System prompt (percent-encoded)

Table 1

Boolean values MUST be represented as "true" or "false" (case-insensitive). Numeric values MUST be represented in decimal notation.

Provider-specific parameters (e.g., "frequency_penalty", "presence_penalty", "cacheControl") SHOULD use the parameter names defined by the provider's API documentation and MUST be percent-encoded if they contain reserved characters.

7. Examples

Basic usage with a cloud provider:

llm://api.openai.com/gpt-5?temp=0.7&max_tokens=1500

With authentication:

llm://my-app:sk-proj-abc123@api.openai.com/gpt-5?temp=0.7

Self-hosted model (Ollama):

```
llm://localhost:11434/llama3
```

Anthropic via aggregator (Bedrock):

```
llm://bedrock.us-west-2.amazonaws.com/anthropic/claude-sonnet-4.5
  ?temp=0.8&cache=true
```

Vercel AI SDK-style namespacing:

```
llm://sdk.vercel.ai/anthropic/claude-sonnet-4.5
  ?temp=0.8&web_search=%7B%22maxUses%22%3A3%7D
```

Local model with JSON output:

```
llm://localhost:11434/mistral?format=json&temp=0
```

With fragment for application context:

```
llm://api.anthropic.com/claude-sonnet-4.5
  ?temp=0.3&max_tokens=4096#code-review
```

8. Encoding Considerations

LLM URIs MUST conform to the encoding rules defined in [RFC3986] Section 2. In particular:

* The model identifier in the path component MUST percent-encode any characters not in the "unreserved" set or "/" delimiter.

* Query parameter values MUST percent-encode reserved characters per Section 2.2 of [RFC3986].

* The userinfo component MUST percent-encode ":" in API keys that contain literal colons, and "@" in any subcomponent.

* Structured values (JSON objects) in query parameters MUST be fully percent-encoded.

Implementations SHOULD produce URIs using UTF-8 encoding for any non-ASCII characters, consistent with [RFC3987] IRI-to-URI mapping.

9. Interoperability Considerations

The "llm" scheme is designed to be consumed by LLM client libraries, CLI tools, middleware routers, and orchestration frameworks. It is NOT designed to be resolved by web browsers or general-purpose HTTP clients.

Known interoperability considerations include:

* *Provider API Variance:* LLM providers use different parameter names for equivalent concepts (e.g., "temperature" vs "temp", "max_tokens" vs "maxOutputTokens"). Client libraries SHOULD map the well-known parameter names defined in Section 6 to the provider's expected parameter names.

* *Model Name Formats:* Some providers use simple names ("gpt-5"), others use versioned names ("gpt-5-2025-01-15"), and aggregator services use namespaced paths ("anthropic/claude-sonnet-4.5"). The path component accommodates all of these formats.

* *Authentication Methods:* While this scheme supports credential embedding via userinfo, some providers require additional authentication mechanisms (e.g., OAuth 2.0 tokens, IAM roles, mTLS certificates) that cannot be represented in a URI. Clients MAY use query parameters or out-of-band configuration for such mechanisms.

10. Security Considerations

The "llm" URI scheme presents several security considerations that implementations MUST address.

10.1. Credential Exposure

LLM URIs MAY contain API keys in the userinfo component. This creates a risk of credential leakage through:

* Application logs and monitoring systems.

* Browser history and bookmarks (if URIs are inadvertently used in web contexts).

* Shell history files.

* Error messages and stack traces.

* Version control systems (.env files committed to repositories).

Implementations that produce, store, or transmit LLM URIs MUST:

* Sanitize the userinfo component before logging, by replacing the api-key portion with a redaction marker (e.g., "***").

* Treat LLM URIs containing credentials with the same care as any other bearer token or secret.

Implementations SHOULD support credential resolution via environment variables or secret managers as an alternative to embedding credentials directly in the URI.

10.2. Transport Security

Clients MUST use HTTPS (TLS 1.2 or later, TLS 1.3 RECOMMENDED) for all remote provider communication. Clients MUST NOT send credentials or inference requests over unencrypted HTTP to non-loopback addresses.

For localhost and loopback addresses (127.0.0.0/8, ::1), clients MAY use HTTP, as this is common for self-hosted model servers.

10.3. Injection Attacks

Because query parameters are passed to provider APIs, implementations MUST validate and sanitize all parameter values before including them in API requests. In particular:

* Numeric parameters MUST be validated as numeric.

* Boolean parameters MUST be validated as boolean.

* String parameters MUST be bounded in length.

* Structured values (JSON) MUST be parsed and validated before forwarding.

10.4. Prompt Injection via URI

The "system" query parameter, if supported, carries a percent-encoded system prompt. This creates a vector for prompt injection if URIs are constructed from untrusted input. Implementations SHOULD treat the "system" parameter as untrusted input and apply appropriate content filtering.

10.5. Server-Side Request Forgery (SSRF)

Applications that accept LLM URIs from untrusted sources (e.g., user-submitted bot configurations, webhook endpoints, or plugin manifests) are vulnerable to Server-Side Request Forgery. An attacker could supply a URI such as llm://localhost:8080/admin or llm://169.254.169.254/latest/meta-data to cause the client library to issue HTTP requests to internal services, cloud metadata endpoints, or other resources that should not be externally reachable.

Implementations that accept LLM URIs from untrusted input MUST:

* Maintain an allowlist of permitted provider hosts, OR maintain a blocklist that includes at minimum: loopback addresses (127.0.0.0/8, ::1), link-local addresses (169.254.0.0/16, fe80::/10), private network ranges (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16), and cloud metadata endpoints.

* Resolve hostnames and validate the resulting IP address against the allowlist/blocklist BEFORE establishing a connection, to prevent DNS rebinding attacks.

* Reject URIs with IP address literals in the host component when operating in production environments, unless explicitly configured to allow them.

In production deployments, clients SHOULD default to blocking loopback and private network addresses. Localhost access SHOULD only be permitted when explicitly enabled for local model serving (e.g., Ollama).

11. Privacy Considerations

LLM URIs may reveal information about:

* Which AI providers and models a user or organization employs.

* Inference configuration choices that may indicate use case (e.g., low temperature for code generation, high temperature for creative writing).

* Application identifiers embedded in the userinfo app-label.

When LLM URIs are shared, logged, or transmitted, this metadata leakage SHOULD be considered. Implementations SHOULD provide mechanisms to strip or redact URI components before sharing.

12. IANA Considerations

12.1. URI Scheme Registration: llm

In accordance with [RFC7595], this section provides the registration request for the "llm" URI scheme.

Scheme name: llm

Status: Provisional

Applications/protocols that use this scheme name: LLM client libraries, AI application frameworks, CLI tools, and orchestration systems that connect to Large Language Model inference APIs.

Contact: Dan Levy <dan@danlevy.net>

Change controller: Dan Levy <dan@danlevy.net>

References: This document; RFC 3986; RFC 7595.

12.2. Well-Known Query Parameters Registry

This document requests IANA create a new registry titled "LLM URI Scheme Well-Known Query Parameters" under the "Uniform Resource Identifier (URI) Schemes" group.

The initial contents of the registry are the parameters defined in Section 6 of this document. New entries may be added via Specification Required [RFC8126] policy.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/rfc/rfc7595>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

13.2. Informative References

[LLM-CONN] Levy, D., "It’s Time for LLM Connection Strings", January 2026, <https://danlevy.net/llm-connection-strings/>.

[OPENAI-API] OpenAI, "API Reference", n.d., <https://platform.openai.com/docs/api-reference>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <https://www.rfc-editor.org/rfc/rfc3987>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

Appendix A. Comparison with Database Connection URIs

The following table illustrates the structural analogy between the "llm" scheme and established database connection URI schemes:

Component	Database URI (postgres)	LLM URI (llm)
Scheme	postgres://	llm://
Auth	user:password	app-label:api-key
Host	db.example.com	api.openai.com
Port	:5432	:443 (default)
Resource	/database_name	/model-name
Options	?sslmode=require	?temp=0.7&max_tokens=1500

Table 2

This structural equivalence means that existing URI parsing libraries, URL-safe storage mechanisms, and secret management tools that already handle database URIs can be extended to handle LLM URIs with minimal effort.

Acknowledgements

The concept of LLM connection strings was first described in "It’s Time for LLM Connection Strings" [LLM-CONN] published at danlevy.net. The author thanks Yakov Shafranovich for suggesting IANA registration and the commenters who provided early feedback.

Author’s Address

Dan Levy
DanLevy Consulting
Denver, CO,
United States of America
Email: dan@danlevy.net
URI: https://danlevy.net