Inference Providers integration requirements 💟

We shipped https://huggingface.co/blog/inference-providers recently with an initial set of partners, and it's been quite successful so far. We'd love to have you on board next 🔥
This doc lists the integration requirements. Let us know if you need any help!

1. Play with the huggingface.js/inference client

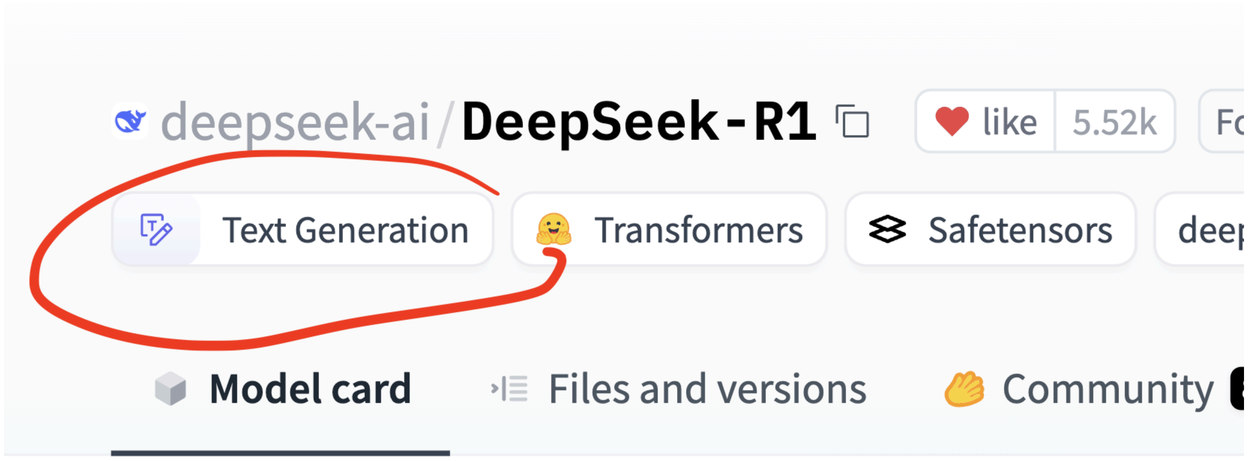
```
Unset
Note: if you provide inference only for LLM/VLMs following the OpenAI API, you
can probably skip this section
```

The first step to understand the integration is to take a look at the JS inference client that lives inside the huggingface.js repo:
https://github.com/huggingface/huggingface.js/tree/main/packages/inference

This is the client that powers our Inference widgets on model pages, and is the blueprint implementation for other SDKs like Python's huggingface_hub and other tools.

What is a Task

You will see that inference methods (textToImage, chatCompletion, etc) have names that closely mirror the task names. `task`, also known as pipeline_tag in the HF ecosystem, is the type of model (basically which types of inputs and outputs the model has), for instance “text-generation” or “text-to-image”. It is indicated prominently on model pages, here:



The list of all possible tasks is https://huggingface.co/tasks and the list of method names is in the README at https://github.com/huggingface/huggingface.js/tree/main/packages/inference

‘chatCompletion’ is an exception as it is not a pipeline_tag. It includes models with either pipeline_tag=“text-generation” or pipeline_tag=“image-text-to-text” and tagged as “conversational”.

Task API schema

We enforce an API schema for each task type, to make it easier for end users to use different models.
To be compatible, the third-party API must adhere to the "standard" shape API we expect on HF model pages for each pipeline task type.

This is not an issue for LLMs as everyone converged on the OpenAI API anyways, but can be more tricky for other tasks like "text-to-image" or "automatic-speech-recognition" where there exists no standard API.

For ex: you can find the expected schema for Text to Speech here: https://github.com/huggingface/huggingface.js/blob/0a690a14d52041a872dc103846225603599f4a33/packages/tasks/src/tasks/text-to-speech/spec/input.json#L4 (and similarly for other supported tasks)

If your API for a given task is different from HF's, it is not an issue: you can tweak the code in huggingface.js to be able to call your models, ie. provide some kind of "translation" of parameter names and output names.

Run the JS code and add some tests to make sure it works well. We can help with this step!

2. Implement the Model Mapping API

Once you've verified the huggingface.js/inference client can call your models successfully, you can use our Model Mapping API.

This API lets a Partner "register" that they support model X, Y or Z on HF.

This enables:
- the inference widget on corresponding model pages,
- inference compatibility throughout the HF ecosystem (Python and JS client SDKs for instance), and any downstream tool or library.

It replaces the initial hardcoded mappings in huggingface.js: for instance https://github.com/huggingface/huggingface.js/blob/e288dbe23c89962dae38a2d7caee5c4fd5a71e4e/packages/inference/src/providers/together.ts

Register a mapping item

```
Unset
POST /api/partners/{provider}/models
```

Create a new mapping item, with the following body (JSON-encoded):

```
JavaScript

{
	task: WidgetType,	// required
	hfModel?: string,	// required in most cases, see hfFilter below for exception
	hfFilter?: string[], // (both can't be defined at the same time)
	// ^Power user move: register a "tag" slice of HF in one go.
	// Example: tag == "base_model:adapter:black-forest-labs/FLUX.1-dev" for all Flux-dev LoRAs
```

```
    providerModel: string // required: the partner's "model id" i.e. id on
your side
}
```

task, also known as pipeline_tag in the HF ecosystem, is the type of model / type of API (examples: text-to-image, text-generation, but use conversational for chat models)
hfModel is the model id on the Hub's side
providerModel is the model id on your side (can be the same or different).

Access control

You need to be in the {provider} Hub organization (e.g. https://huggingface.co/togethercomputer for TogetherAI) with Write permissions to be able to access this endpoint.

Validation

The endpoints validates that:

- hfModel is indeed of pipeline_tag == task OR task is "conversational" and the model is compatible.
- hfFilter, if specified, has some additional safeguard validation
- (in the future) we auto-test that the Partner's API successfully responds to a huggingface.js/inference call of the corresponding task i.e. the API shape is valid

Delete a mapping item

```
Unset
DELETE /api/partners/{provider}/models
{ hfModel: "" }
# or
{ hfFilter: "" }
# or even!!
{ providerModel: "" }
```

Update a mapping item

Delete it then recreate it.

List the whole mapping

```
Unset
GET /api/partners/{provider}/models
```

This gets all mapping items from the DB. For clarity/DX, the output is grouped by `task`.

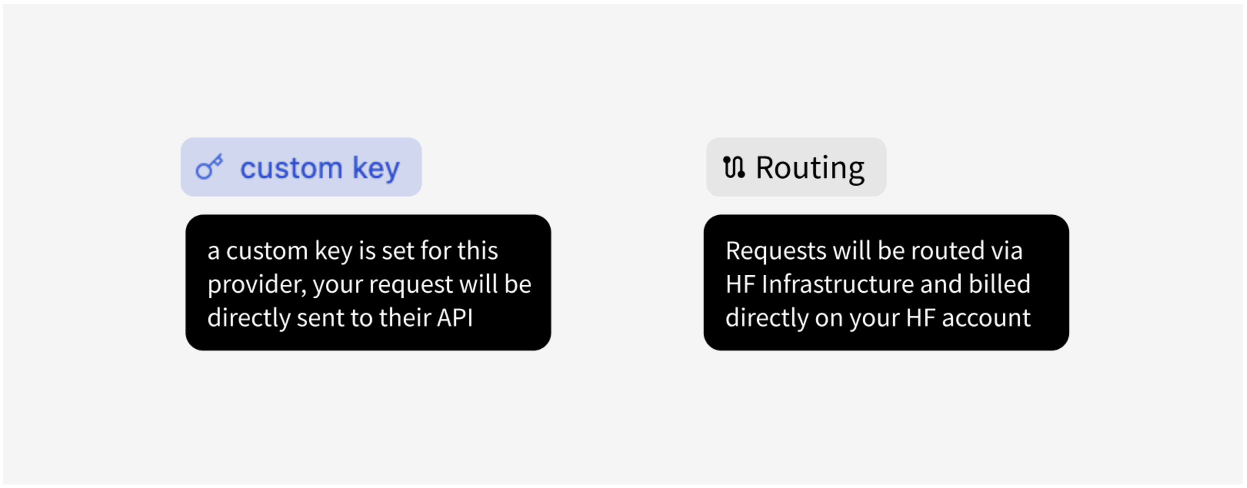
‼ Important Note: this is publicly accessible. It's useful to be transparent by default, helps debug client SDKs, etc.

```javascript
JavaScript
{
	"text-to-image": {
		"black-forest-labs/FLUX.1-Canny-dev":
"black-forest-labs/FLUX.1-canny",
		"black-forest-labs/FLUX.1-Depth-dev":
"black-forest-labs/FLUX.1-depth",
		"filter|base_model:adapter:black-forest-labs/FLUX.1-dev":
"fal/flux-lora",
	},
	"conversational": {
		"deepseek-ai/DeepSeek-R1": "deepseek-ai/DeepSeek-R1",
	},
	"text-generation": {
		"meta-llama/Llama-2-70b-hf": "meta-llama/Llama-2-70b-hf",
		"mistralai/Mixtral-8x7B-v0.1": "mistralai/Mixtral-8x7B-v0.1",
	},
};
```

This outputs the equivalent of the hardcoded mappings we originally used in huggingface.js: https://github.com/huggingface/huggingface.js/blob/e288dbe23c89962dae38a2d7caee5c4fd5a71e4e/packages/inference/src/providers/together.ts

3. For billing, implement a Request-Cost HTTP trailer

For **routed** requests (see figure below), i.e. when users authenticate via HF, our intent is that our users only pay the standard provider API rates. There's no additional markup from us, we just pass through the provider costs directly.


custom key
a custom key is set for this provider, your request will be directly sent to their API
Routing
Requests will be routed via HF Infrastructure and billed directly on your HF account

For LLM providers, the current workaround is to extract numbers of input and output tokens in the responses and multiply by a hardcoded pricing table – this is quite brittle.

We propose an easier way to figure out this cost and charge it to our users, by asking you to provide the cost for each request in an HTTP trailer.

What is a HTTP Trailer

it's like a HTTP header, but comes *after* the response body (given generally you only know at the end of the AI model generation how much it costs):

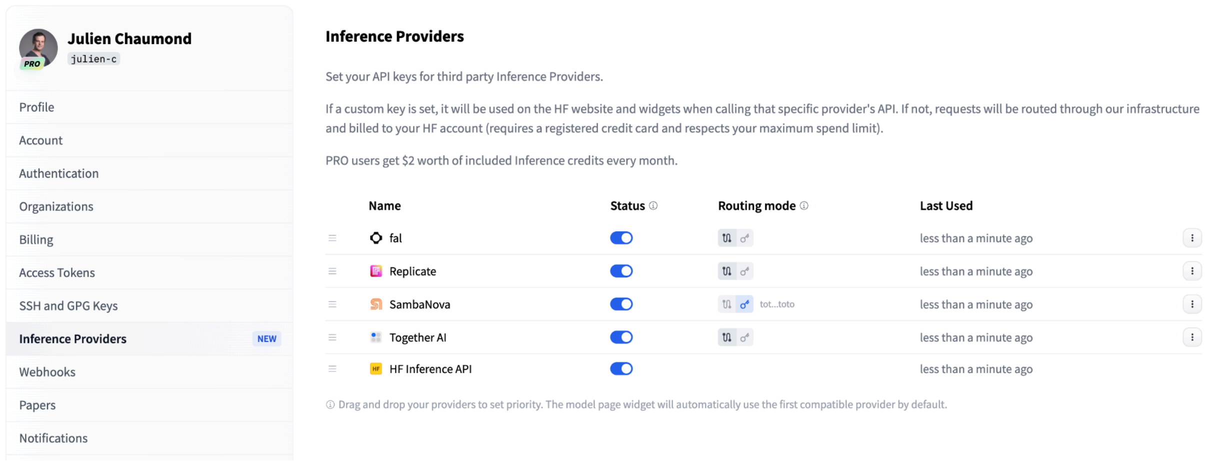
```
Unset
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Trailer: Request-Cost

--- response body ---
Request-Cost: nanousd=120
```

In which unit is it defined

We require the price to be a integer number of nano-USDs (10^-9 USD)

4. Final miscellaneous details

Question: by default in which order do we list providers in the settings page



Julien Chaumond
julien-c
PRO

- Profile
- Account
- Authentication
- Organizations
- Billing
- Access Tokens
- SSH and GPG Keys
- Inference Providers NEW
- Webhooks
- Papers
- Notifications

Inference Providers

Set your API keys for third party Inference Providers.

If a custom key is set, it will be used on the HF website and widgets when calling that specific provider's API. If not, requests will be routed through our infrastructure and billed to your HF account (requires a registered credit card and respects your maximum spend limit).

PRO users get $2 worth of included Inference credits every month.

Name	Status	Routing mode	Last Used
fal	●		less than a minute ago
Replicate	●		less than a minute ago
SambaNova	●	tot...toto	less than a minute ago
Together AI	●		less than a minute ago
HF Inference API	●		less than a minute ago

ⓘ Drag and drop your providers to set priority. The model page widget will automatically use the first compatible provider by default.

Answer: the default sort is by `total number of requests routed by HF over the last 7 days`. This order defines which provider will be used in priority by the widget on the model page (but the user's order takes precedence).