

**IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE**

FRIENDLIAI INC.,

Plaintiff,

v.

HUGGING FACE, INC.,

Defendant.

C.A. No. 1:23-CV-00816-MN

**PLAINTIFF’S FRIENDLIAI INC.’S
FINAL INFRINGEMENT CONTENTIONS**

Pursuant to Section 6(e) of the Scheduling Order (D.I. 17), Plaintiff FriendliAI Inc. (“FriendliAI”) hereby submits its final infringement contentions against Hugging Face, Inc. (“Hugging Face”). These infringement contentions include claim charts that relate each known Hugging Face Accused Product to the currently asserted claims of the Asserted Patents.¹

The claim charts are attached to these disclosures as Exhibit A (for the ’775 patent) and Exhibit B (for the ’520 patent). Specifically, these disclosures contend that the Accused Functionality in each of the Accused Products infringe (1) at least claims 1-18 of the ’775 patent, and (2) at least claims 1-66 of the ’520 patent.²

¹ The Asserted Patents include U.S. Patent No. 11,442,775 (the ’775 patent) and U.S. Patent No. 11,836,520 (the ’520 patent).

² The Accused Products include Text Generation Inference (TGI), and any product, service, platform, solutions, model, dataset, application, or toolkit that Hugging Face has made, sold, offered for sale, advertised, marketed, used, or imported into the United States that incorporates, applies, or otherwise uses Text Generation Inference (TGI), including but not limited to the TGI toolkit distributed or otherwise made available by Hugging Face (including open source and under license), Spaces, Inference Endpoints, Enterprise Hub (formerly known as Private Hub), Hugging Face Hub, HuggingChat, OpenAssistant, and Docker Hub containers, including all past, current, and future versions of these products and services, even if they are not explicitly referred to by

Hugging Face (1) directly infringes by running TGI for its customers on its own servers; (2) directly infringes through internal testing of its products; (3) directly infringes by selling software that allegedly “performs and/or dictates and controls performance of (or automatically performs)” the claimed method steps in the hands of Hugging Face or its customers, (4) indirectly infringes by inducing customers to run TGI, and (5) contributorily infringes by selling and/or distributing a component (TGI) especially designed for use in the patented invention.

FriendliAI makes these disclosures based on discovery received to date. Discovery is ongoing and Hugging Face has continuously delayed providing required discovery in this case. (*See* D.I. 62.) FriendliAI reserves the right to modify, supplement, amend, or otherwise alter these disclosures after receiving and reviewing the source code and pending discovery. Because FriendliAI’s infringement investigation is ongoing, FriendliAI reserves the right to supplement or amend these disclosures as appropriate under the Scheduling Order, the Federal Rules, the Local Rules, the Scheduling Order, and the District of Delaware’s Default Standard for Discovery, and in light of additional fact and/or expert discovery. In addition to the documents cited herein, FriendliAI may rely on additional documents and testimony. Should Hugging Face take the position that any limitation of any claim identified in Exhibits A and B is not present literally in the Accused Products, FriendliAI reserves the right to assert infringement under the doctrine of equivalents.

FriendliAI also makes these disclosures before the parties have finalized claim construction positions, submit claim construction briefing to the Court, or participate in a *Markman* hearing.

these product names, including any that may be released during the pendency of this matter (“Accused Products”). FriendliAI identifies the above Accused Products, but further discovery may be necessary to reveal additional Accused Products. FriendliAI reserves the right to modify, supplement, amend, or otherwise alter these disclosures based on forthcoming discovery.

FriendliAI therefore reserves the right to supplement or amend these disclosures as appropriate in light of ongoing claim construction proceedings and any claim construction ruling by the Court.

Dated: July 1, 2024

Respectfully submitted,

POTTER ANDERSON & CORROON LLP

OF COUNSEL:

Michael J. Sacksteder
Shreyas A. Kale
Samantha Ong
FENWICK & WEST LLP
555 California Street, 12th Floor
San Francisco, CA 94104
Telephone: 415.875.2300
Facsimile: 415.281.1350
MSacksteder@fenwick.com
SKale@fenwick.com
Song@fenwick.com

Jessica M. Kaempf
FENWICK & WEST LLP
401 Union Street, 5th Floor
Seattle, WA 98101
Tel: (206) 389-4550
JKaempf@fenwick.com

By: /s/ Jessica Kaempf
David E. Moore (#3983)
Bindu A. Palapura (#5370)
Andrew L. Brown (#6766)
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19801
Tel: (302) 984-6000
dmoore@potteranderson.com
bpalapura@potteranderson.com
abrown@potteranderson.com

Attorneys for Plaintiff FriendliAI Inc.

CERTIFICATE OF SERVICE

I hereby certify that on July 1, 2024, a true and correct copy of the foregoing document was served via electronic mail upon the following counsel of record:

Stephen J. Krafchik
Christina B. Vavala
POLSINELLI PC
222 Delaware Avenue, Ste 1101
Wilmington, DE 19801
Telephone: (302) 252-0920
Email: skraftschik@polsinelli.com
Email: cvavala@polsinelli.com

Jason A. Wietjes
POLSINELLI PC
2950 N. Harwood St., Ste 2100
Dallas, TX 75201
Telephone: (214) 661-5519
Email: jwietjes@polsinelli.com
Attorneys for Defendant

/s/ Jessica Kaempf
Jessica Kaempf

EXHIBIT A

FriendliAI Inc. v. Hugging Face, Inc.: Final Infringement Chart for U.S. Patent No. 11,442,775

The claim chart below provides an exemplary mapping of an exemplary accused functionality, Text Generation Inference (“TGI” or “Accused Functionality”), to the claims of U.S. Patent No. 11,442,775 (“the ’775 patent”). Discovery is ongoing and this chart may be amended based on information provided on the Accused Products during discovery. This chart is not intended to provide a comprehensive explanation of infringement or identification of Accused Products and Services, which FriendliAI will provide in accordance with the Court’s rules and case schedule. Plaintiff is seeking Defendants’ internal documentation, including without limitation technical documentation, in discovery and therefore Plaintiff reserves the right to amend or further supplement this chart. Proposed claim constructions have not been finalized and claim construction briefing is still pending. The accused functionality meets each claim limitation regardless of claim construction, as further indicated below. Plaintiff, however, reserves the right to amend or further supplement this chart based on the Court’s claim construction. Source code for certain accused products has not yet been bates stamped and provided to Plaintiff, and these contentions may be amended pursuant to that source code.

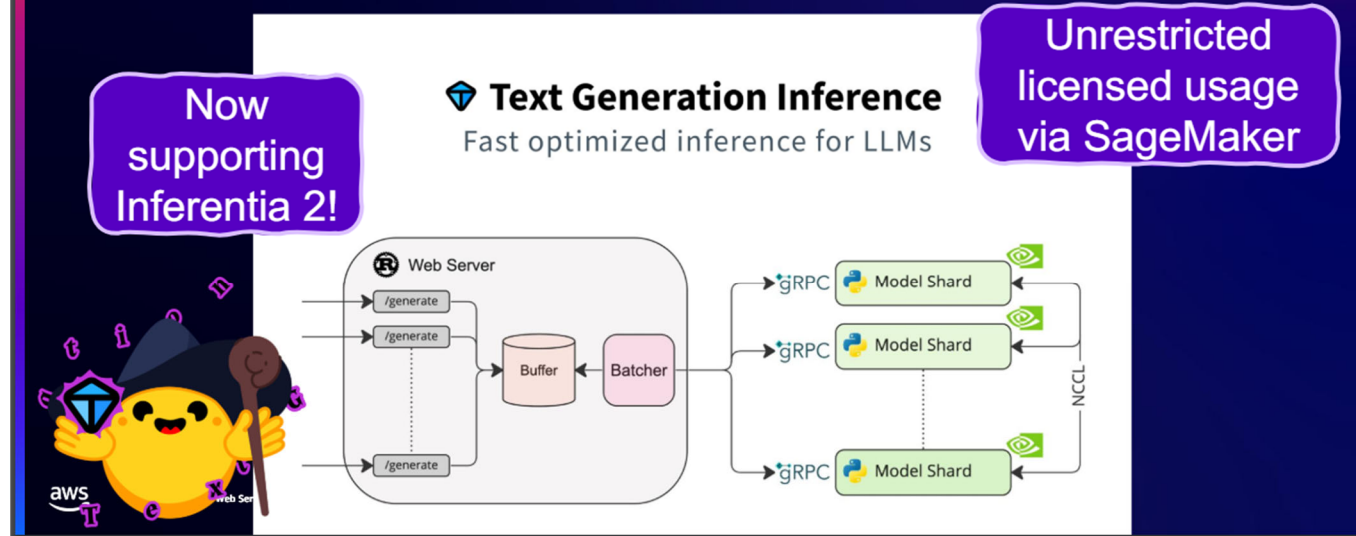
U.S. Patent No. 11,442,775	Accused Functionality
Claim 1. A method of dynamically executing batches of requests on one or more execution engines running a machine-learning transformer model, comprising:	Defendant has designed and developed, and has used, sold, offered for sale, and/or otherwise made available (and continues to use, sell, offer for sale, and/or otherwise make available), the Accused Products and Services that use Text Generation Inference (“TGI” or the “Accused Functionality”). The Accused Functionality performs operations for dynamically executing batches of requests on one or more execution engines running a machine-learning transformer model. See https://huggingface.co/text-generation-inference.¹ The Accused Products include Text Generation Inference (TGI), and any product, service, platform, solutions, model, dataset, application, or toolkit that Hugging Face has made, sold, offered for sale, advertised, marketed, used, or imported into the United States that incorporates, applies, or otherwise uses Text Generation Inference (TGI), including but not limited to the TGI toolkit distributed or otherwise made available by Hugging Face (including open source and under license), Spaces, Inference Endpoints, Enterprise Hub (formerly known as Private Hub), Hugging Face Hub, HuggingChat, OpenAssistant, and Docker Hub containers, including all past, current, and future versions of these products and services, even if they are not explicitly referred to by these product names, including any that may be released during the pendency of this matter (“Accused Products”). FriendliAI identifies the above Accused Products, but further discovery may be necessary to reveal additional Accused Products. FriendliAI reserves the right to modify, supplement, amend, or otherwise alter these disclosures based on forthcoming discovery. Hugging Face indirectly infringes the Asserted Patents through contributory infringement and induced infringement. Hugging Face had knowledge of the Asserted Patents (<i>see</i> Amended Compl., at, e.g., ¶¶ 43, 50, 53, 63, 66). Hugging Face contributorily infringes each of the Asserted Patents by selling and/or distributing a component especially designed for use in the patented invention. Specifically, Hugging Face sells and/or distributes TGI. Further, TGI is (at a minimum) a material component of the claimed invention, and is not suitable for substantial non-infringing uses. The batching efficiencies that Hugging Face touts in its description of TGI are at the core of many of the Hugging Face products at issue in this case. Hugging Face induces infringement of the Asserted Patents by actively and knowingly aiding and abetting Hugging Face users’ and business partners’ direct infringement. Hugging Face had knowledge of

¹ By charting the preamble, Plaintiff does not take any position as to whether the preamble is a limitation of the claim.

	the infringement because it continued to use and promote technology infringing the Asserted Patents since at least July 20, 2023, when it was placed on notice that the Accused Products infringed the claims of the asserted patents. FriendlyAI further incorporates by reference the allegations in the Amended Complaint, at, e.g., ¶¶ 41-42. Hugging Face provides users and business partners with instructions on how to use TGI in an infringing manner. Hugging Face provides tools and instructions on its website on how to use TGI to perform steps that result in the directly infringing use. Hugging Face also advertises or promotes the use of TGI. Additionally, Hugging Face is responsible for and exercises control over the manufacturing and development of TGI. Hugging Face has actively encouraged its customers to commit direct infringement of the asserted claims with knowledge that the acts encouraged constitute infringement and a specific intent to infringe the asserted claims. The Accused Products also indirectly infringe at least because Hugging Face provides TGI to customers to execute on their own servers, Hugging Face induces customers to infringe the '775 patent claims by providing TGI and instructions to execute TGI on third-party systems, or Hugging Face indirectly infringes by providing TGI in any other fashion to be used with other software or hardware. (<i>See</i> https://github.com/huggingface/text-generation-inference). Hugging Face also indirectly infringes when it provides enterprise consulting and support to customers to implement TGI on either the customer's own hardware or in their own particular environment. (<i>See</i> DEF_000298; FRIENDLIAI_00001163). In some cases, this includes Hugging Face providing custom TGI containers optimized to execute on the customer's hardware or in the customer's environment. <i>Id.</i> Hugging Face also indirectly infringes when it licenses or otherwise provides TGI to partners and collaborators to execute in their environment, and encourages users to use TGI in their partners' environment. (<i>See</i> DEF_360181; DEF_360159; FRIENDLIAI_00001163).
--	--

Cost effective: With Text Generation Inference

TGI is designed for maximum throughput for LLM inference



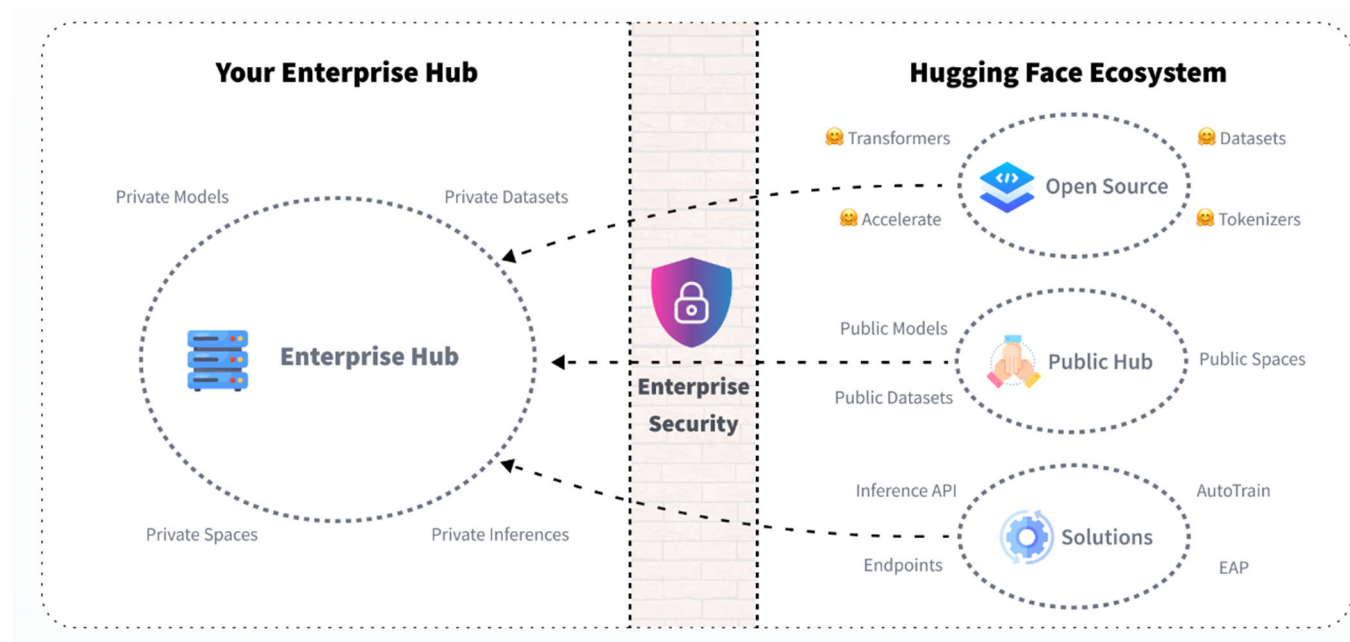
(See <https://www.youtube.com/watch?v=cTyXDhiltXw>; see also https://d1.awsstatic.com/events/Summits/reinvent2023/AIM328_Accelerate-FM-development-with-Amazon-SageMaker-JumpStart.pdf).

Hugging Face's Spaces product infringes the claims of the '775 patent by incorporating TGI into its execution environment: DEF_347045; DEF_347048; DEF_347057; DEF_347071; https://huggingface.co/docs/text-generation-inference/en/basic_tutorials/consuming_tgi ("ChatUI can automatically consume the TGI server and even provides an option to switch between different TGI endpoints. You can try it out at Hugging Chat, or use the ChatUI Docker Space to deploy your own Hugging Chat to Spaces."); https://huggingface.co/spaces/ysharma/Explore_llamav2_with_TGI. Customers are encouraged to create their own applications on Spaces (e.g., Chatbots) using TGI. (See

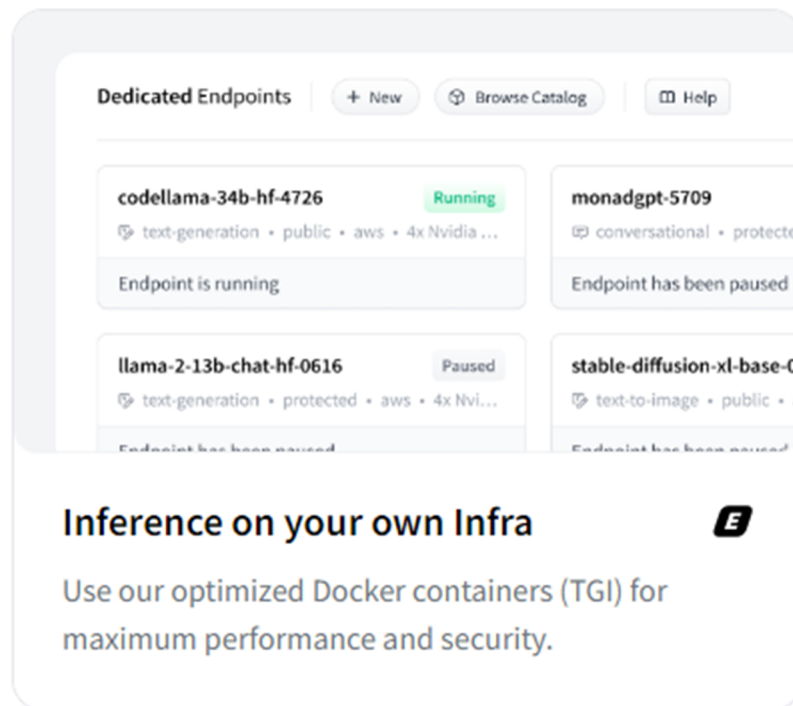
<https://github.com/huggingface/chat-ui>; <https://huggingface.co/chat/>; DEF_000050;
https://huggingface.co/docs/text-generation-inference/en/basic_tutorials/consuming_tgi).

Hugging Face's Inference Endpoints product infringes the claims of the '775 patent by incorporating TGI into its execution environment: <https://huggingface.co/blog/inference-endpoints-llm>; <https://huggingface.co/text-generation-inference>. Hugging Face provides a custom TGI container to users of Inference Endpoints that allows users to easily deploy TGI onto various hardware systems. Hugging Face builds and deploys TGI for users of Inference Endpoints. (DEF_347031; DEF_347040; DEF_347045; DEF_347074; DEF_347080; DEF_347083; DEF_347085; DEF_347112-DEF_347116; DEF_347040).

Hugging Face's Enterprise Hub product infringes the claims of the '775 patent by incorporating TGI into its execution environment:



<https://huggingface.co/enterprise>; see <https://huggingface.co/text-generation-inference>.



(<https://huggingface.co/enterprise>; see also FRIENDLIAI_00001163. Enterprise Hub implements TGI for its customers, as evident from its source code. (DEF_30711-DEF_30722; DEF_30726; DEF_30728; DEF_30730- DEF_30732; DEF_30787; DEF_30797; DEF_30800). The Hugging Face Hub infringes for at least the same reasons as the Enterprise Hub, as the source code overlaps for the two products.

Hugging Face’s HuggingChat product infringes the claims of the ’775 patent by incorporating TGI into its execution environment: <https://huggingface.co/text-generation-inference>; <https://huggingface.co/chat/>; <https://huggingface.co/docs/text-generation-inference/index> (“Text Generation Inference is used in production by multiple projects, such as: Hugging Chat, an open-source interface for open-access models, such as Open Assistant and Llama.”).

	Hugging Face deploys the OpenAssistant product that infringes the claims of the '775 patent by incorporating TGI into its execution environment: https://huggingface.co/text-generation-inference ("Text Generation Inference is already used by customers such as IBM, Grammarly, and the Open-Assistant initiative implements optimization for all supported model architectures"); https://huggingface.co/chat/; https://huggingface.co/docs/text-generation-inference/index ("Text Generation Inference is used in production by multiple projects, such as: Hugging Chat, an open-source interface for open-access models, such as Open Assistant and Llama. OpenAssistant, an open-source community effort to train LLMs in the open"). Hugging Face's Docker Hub product infringes the claims of the '775 patent by incorporating TGI into its execution environment and by distributing TGI as a Docker Container: https://huggingface.co/docs/text-generation-inference/quicktour ("Quick Tour - The easiest way of getting started is using the official Docker container."); https://huggingface.co/docs/text-generation-inference/installation ("This section explains how to install the CLI tool as well as installing TGI from source. The strongly recommended approach is to use Docker, as it does not require much setup. Check the Quick Tour to learn how to run TGI with Docker."); https://github.com/huggingface/text-generation-inference.
1[A] receiving, by a serving system, one or more requests for execution, the serving system including a scheduler and one or more execution engines each coupled to access a machine-learning transformer model including at least a set of decoders;	The Accused Functionality performs the step of receiving, by a serving system, one or more requests for execution, the serving system including a scheduler and one or more execution engines each coupled to access a machine-learning transformer model including at least a set of decoders. The Accused Functionality performs the step of receiving, by a serving system, one or more requests for execution. The Accused Functionality receives a request for execution:

```

33 /// Generate tokens if 'stream == false' or a stream of token if 'stream == true'
34 #![utoipa::path(
35     post,
36     tag = "Text Generation Inference",
37     path = "/",
38     request_body = CompatGenerateRequest,
39     responses(
40         (status = 200, description = "Generated Text",
41             content(
42                 ("application/json" = GenerateResponse),
43                 ("text/event-stream" = StreamResponse),
44             )),
45         (status = 424, description = "Generation Error", body = ErrorResponse,
46             example = json ! ({"error": "Request failed during generation"})),
47         (status = 429, description = "Model is overloaded", body = ErrorResponse,
48             example = json ! ({"error": "Model is overloaded"})),
49         (status = 422, description = "Input validation error", body = ErrorResponse,
50             example = json ! ({"error": "Input validation error"})),
51         (status = 500, description = "Incomplete generation", body = ErrorResponse,
52             example = json ! ({"error": "Incomplete generation"})),
53     )
54 )]]
55 #![instrument(skip(infer, req))]
56 ✓ async fn compat_generate(
57     default_return_full_text: Extension<bool>,
58     infer: Extension<Infer>,
59     req: Json<CompatGenerateRequest>,
60 ) -> Result<Response, (StatusCode, Json<ErrorResponse>)> {
61     let mut req = req.0;
62
63     // default return_full_text given the pipeline_tag
64     if req.parameters.return_full_text.is_none() {
65         req.parameters.return_full_text = Some(default_return_full_text.0)
66     }
67
68     // switch on stream
69     if req.stream {
70         Ok(generate_stream(infer, Json(req.into())))
71             .await
72             .into_response()
73     } else {
74         let (headers, generation) = generate(infer, Json(req.into())).await?;
75         // wrap generation inside a Vec to match api-inference
76         Ok((headers, Json(vec![generation.0])).into_response())
77     }
78 }

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/server.rs#L33-L78>).

The Accused Functionality includes a scheduler:

```

239 // Batching logic
240 // Will be launched in a background Tokio task
241 //
242 // Batches requests and sends them to the inference server
243 async fn batching_task(
244     mut client: ShardedClient,
245     waiting_served_ratio: f32,
246     max_batch_total_tokens: u32,
247     max_waiting_tokens: usize,
248     queue: Queue,
249     shared: Arc<Shared>,
250     generation_health: Arc<AtomicBool>,
251 ) {
252     // Infinite loop
253     loop {
254         // Wait for a notification from the Infer struct
255         shared.batching_task.notified().await;
256
257         // Get the next batch from the queue
258         // This batch might be smaller than the maximum batch size if there are not enough requests
259         // waiting in the queue
260         while let Some((mut entries, batch, span)) =
261             queue.next_batch(None, max_batch_total_tokens).await
262         {

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L239-L348>; see also <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/server.rs#L580-L589>)

The Accused Functionality scheduler puts the received requests into a queue:

```

115 // Append the request to the queue
116 self.queue.append(Entry {
117     request: valid_request,
118     response_tx,
119     span: Span::current(),
120     temp_span: None,
121     queue_time: Instant::now(),
122     batch_time: None,
123 });
124
125 // Notify the background task that we have a new entry in the queue that needs
126 // to be batched
127 self.shared.batching_task.notify_one();

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L115-L127>; see also <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L239-L348>)

The Accused Functionality includes one or more one or more execution engines, comprising one or more shards:

```
987     spawn_shards(  
988         num_shard,  
989         &args,  
990         shutdown.clone(),  
991         &shutdown_receiver,  
992         shutdown_sender,  
993         &status_receiver,  
994         status_sender,  
995         running.clone(),  
996     );  
---
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/launcher/src/main.rs#L987-L996>; see also https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L97-L155; see also https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/cli.py#L154-L155).

Each of the execution engines in the Accused Functionality is coupled to access a machine-learning transformer model including at least a set of decoders. The Accused Functionality gets a model:

```
123     try:  
124         model = get_model(model_id, revision, sharded, quantize, trust_remote_code)  
125     except Exception:  
126         logger.exception("Error when initializing model")  
127         raise
```

(https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L123-L124).

The machine-learning transformer models used by the Accused Functionality include at least a set of decoders:

	<pre> 155 if model_type == "bloom": 156 return BLOOMSharded(157 model_id, revision, quantize=quantize, trust_remote_code=trust_remote_code 158) 159 160 elif model_type == "gpt_neox": 161 if FLASH_ATTENTION: 162 return FlashNeoXSharded(163 model_id, 164 revision, 165 quantize=quantize, 166 trust_remote_code=trust_remote_code, 167) 168 elif sharded: 169 return GPTNeoXSharded(170 model_id, 171 revision, 172 quantize=quantize, 173 trust_remote_code=trust_remote_code, 174) 175 else: 176 return CausalLM(177 model_id, 178 revision, 179 quantize=quantize, 180 trust_remote_code=trust_remote_code, 181) 182 183 elif model_type == "llama": 184 if FLASH_ATTENTION: 185 return FlashLlama(186 model_id, 187 revision, 188 quantize=quantize, 189 trust_remote_code=trust_remote_code, 190) </pre> (https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/_init_.py#L98-L276).
1[B] scheduling, by the scheduler, a batch of requests including the	The Accused Functionality performs the step of scheduling, by the scheduler, a batch of requests including the one or more requests for execution on an execution engine.

one or more requests for execution on an execution engine;

The Accused Functionality performs the scheduling of a batch of requests, including the one or more requests for execution on an execution engine. The Accused Functionality removes a batch of requests, comprising the one or more requests, from the scheduling queue, then schedules execution of the first iteration on the execution engine:

```
252     // Infinite loop
253     loop {
254         // Wait for a notification from the Infer struct
255         shared.batching_task.notified().await;
256
257         // Get the next batch from the queue
258         // This batch might be smaller than the maximum batch size if there are not enough requests
259         // waiting in the queue
260         while let Some((mut entries, batch, span)) =
261             queue.next_batch(None, max_batch_total_tokens).await
262         {
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L260-L261>).

```
263         let mut cached_batch = prefill(&mut client, batch, &mut entries, &generation_health)
264             .instrument(span)
265             .await;
266         let mut waiting_tokens = 1;
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L263-L265>).

The Accused Functionality also schedules execution of further iterations on the execution engine:

	<pre> 339 cached_batch = decode(&mut client, batches, &mut entries, &generation_health) 340 .instrument(next_batch_span) 341 .await; 342 waiting_tokens += 1; 343 } </pre> (https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L339-L341).
1[C] generating, by the execution engine, a first set of output	The Accused Functionality performs the step of generating, by the execution engine, a first set of output tokens by applying the transformer model to a first set of inputs for the batch of requests, wherein

tokens by applying the transformer model to a first set of inputs for the batch of requests, wherein applying the transformer model comprises applying at least one batch operation to one or more input tensors associated with the batch of requests;

applying the transformer model comprises applying at least one batch operation to one or more input tensors associated with the batch of requests.

The Accused Functionality first applies the transformer model to a first set of inputs for the batch of requests, wherein applying the transformer model comprises applying at least one batch operation to one or more input tensors associated with the batch of request:

```
361         match client.prefill(batch).await {  
362             Ok((generations, next_batch)) => {
```

- (<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L361-L362>; see also https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L147; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L219; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L239; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L347; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L277-L278).

The Accused Functionality then generates tokens from applying the transformer model to the set of inputs in the batch:

```
90      /// Generate one token for each request in the given batch
91      ///
92      /// Returns Generation for each request in batch
93      /// and the next cached batch
94      #[instrument(skip_all, fields(id = &batch.id, size = &batch.size))]
95 ✓ pub async fn prefill(
96     &mut self,
97     batch: Batch,
98 ) -> Result<(Vec<Generation>, Option<CachedBatch>)> {
99     let futures: Vec<_> = self
100         .clients
101         .iter_mut()
102         .map(|client| Box::pin(client.prefill(batch.clone())))
103         .collect();
```

(https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/sharded_client.rs#L98-L103).

```
97      /// Generate one token for each request in the given batch
98      ///
99      /// Returns Generation for each request in batch
100      /// and the next cached batch
101      #[instrument(skip_all, fields(id = &batch.id, size = &batch.size))]
102 ✓ pub async fn prefill(
103     &mut self,
104     batch: Batch,
105 ) -> Result<(Vec<Generation>, Option<CachedBatch>)> {
106     let request = tonic::Request::new(PrefillRequest { batch: Some(batch) }).inject_context();
107     let response = self.stub.prefill(request).await?.into_inner();
108     Ok((response.generations, response.batch))
109 }
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/client.rs#L107-L108>).

The generated tokens are passed back to the Accused Functionality server, and another batch can be scheduled to execute:

```
61     generations, next_batch = self.model.generate_token(batch)
62     self.cache.set(next_batch)
```

(https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L61-L62).

In order to perform these steps, TGI creates the first batch and adds the input tokens to the batch. (See https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L57-L59; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L93-L99; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L147; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L219; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L239).

The Accused Functionality performs at least one batch operation on the input tensors. (See https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L347; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L277-L278; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L140).

1[D] receiving, by a request processor, a new request from a client device, the new request including a sequence of input tokens;	The Accused Functionality performs the step of receiving, by a request processor, a new request from a client device, the new request including a sequence of input tokens. Defendant has proposed a construction of “a new request” that means “a request different from each of the one or more requests of the [first] batch of requests.” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction. The Accused Functionality receives, by a request processor, a new request from a client device, the new request including a sequence of input token:

```

33 /// Generate tokens if `stream == false` or a stream of token if `stream == true`
34 #[utoipa::path(
35     post,
36     tag = "Text Generation Inference",
37     path = "/",
38     request_body = CompatGenerateRequest,
39     responses(
40         (status = 200, description = "Generated Text",
41             content(
42                 ("application/json" = GenerateResponse),
43                 ("text/event-stream" = StreamResponse),
44             )),
45         (status = 424, description = "Generation Error", body = ErrorResponse,
46             example = json ! ({"error": "Request failed during generation"})),
47         (status = 429, description = "Model is overloaded", body = ErrorResponse,
48             example = json ! ({"error": "Model is overloaded"})),
49         (status = 422, description = "Input validation error", body = ErrorResponse,
50             example = json ! ({"error": "Input validation error"})),
51         (status = 500, description = "Incomplete generation", body = ErrorResponse,
52             example = json ! ({"error": "Incomplete generation"})),
53     )
54 )]
55 #[instrument(skip(infer, req))]
56 async fn compat_generate(
57     default_return_full_text: Extension<bool>,
58     infer: Extension<Infer>,
59     req: Json<CompatGenerateRequest>,
60 ) -> Result<Response, (StatusCode, Json<ErrorResponse>)> {
61     let mut req = req.0;
62
63     // default return_full_text given the pipeline_tag
64     if req.parameters.return_full_text.is_none() {
65         req.parameters.return_full_text = Some(default_return_full_text.0)
66     }
67
68     // switch on stream
69     if req.stream {
70         Ok(generate_stream(infer, Json(req.into())))
71             .await
72             .into_response()
73     } else {
74         let (headers, generation) = generate(infer, Json(req.into())).await?;
75         // wrap generation inside a Vec to match api-inference
76         Ok((headers, Json(vec![generation.0])).into_response())
77     }
78 }

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/server.rs#L33-L78>).

The Accused Functionality then tokenizes the input text into a sequence of input tokens:

```

97     batch_tokenized_inputs = tokenizer(
98         batch_inputs, truncation=True, max_length=max_truncation
99     )["input_ids"]

```

	(https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flan_causal_lm.py#L97-L99).
1[E] scheduling, by the scheduler, a second batch of requests additionally including the new request for execution on the execution engine, the	The Accused Functionality performs the step of scheduling, by the scheduler, a second batch of requests additionally including the new request for execution on the execution engine, the second batch of requests scheduled responsive to determining that the execution engine has memory available to execute the second batch of requests, wherein in a second set of inputs for the second batch of requests, a length of the sequence of input tokens for the new request is different from a length of an input for at least one request other than the new request.

second batch of requests scheduled responsive to determining that the execution engine has memory available to execute the second batch of requests, wherein in a second set of inputs for the second batch of requests, a length of the sequence of input tokens for the new request is different from a length of an input for at least one request other than the new request; and

Defendant has proposed a construction of “second batch of requests” that means “a batch that includes at least one request from the [first] batch of requests.” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction.

Defendant has proposed a construction of “a length of the sequence of input tokens for the new request is different from a length of an input for [the] at least one request [other than the new request]” where “‘length’ refers to a ‘number or quantity of discrete <units>’ for a given context— e.g., the number of ‘tokens’ in a sequence, such as ‘a [number or quantity of discrete tokens] of the sequence of input tokens for the new request is different from a [number or quantity of discrete tokens] of an input for at least one request.’” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction.

The Accused Functionality performs the step of scheduling, by the scheduler, a second batch of requests additionally including the new request for execution on the execution engine. The Accused Functionality receives a new set of requests including the new request:

```
289 // Try to get a new batch
290 if let Some((mut new_entries, new_batch, span)) =
291     queue.next_batch(min_size, token_budget).await
---
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L289-L291>).

The Accused Functionality merges the new set of requests into the previous, first batch of requests:

```
318 // Extend current batch with the new batch
319 if let Some(new_cached_batch) = new_cached_batch {
320     entries.extend(new_entries);
321     batches.push(new_cached_batch);
322 }
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L318-L322>).

The Accused Functionality then schedules a second batch of requests that also includes the new request for execution:

```
339         cached_batch = decode(&mut client, batches, &mut entries, &generation_health)
340         .instrument(next_batch_span)
341         .await;
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L339-L341>).

The Accused Functionality schedules the second batch of requests responsive to determining that the execution engine has memory available to execute the second batch of requests:

```
287         let token_budget = max_batch_total_tokens - batch_max_tokens;
288
289         // Try to get a new batch
290         if let Some((mut new_entries, new_batch, span)) =
291             queue.next_batch(min_size, token_budget).await
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L287-L291>).

```

187         if (prefill_tokens + decode_tokens) > token_budget {
188             // Entry is over budget
189             // Add it back to the front
190             self.entries.push_front((id, entry));
191             break;
192         }

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/queue.rs#L187-L192>).

```

156     /// Overall this number should be the largest possible amount that fits the
157     /// remaining memory (after the model is loaded). Since the actual memory overhead
158     /// depends on other parameters like if you're using quantization, flash attention
159     /// or the model implementation, text-generation-inference cannot infer this number
160     /// automatically.
161     #[clap(default_value = "32000", long, env)]
162     max_batch_total_tokens: u32,

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/launcher/src/main.rs#L156-L162>).

The Accused Functionality performs no check to validate that the inputs for the second batch of requests are the same length, and implies that requests in a batch can have an arbitrary length. (See <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/queue.rs>; <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs>).

TGI also confirms this behavior by explaining that it performs continuous batching, which allows for new requests, of different length than the first request, to be processed in the same batch:

Text Generation Inference

Text Generation Inference is a production ready inference container developed by Hugging Face to enable easy deployment of large language models.

Its main features are:

- Continuous batching
- Token streaming using Server-Sent Events (SSE)
- Tensor Parallelism for faster inference on multiple GPUs
- Optimized transformers code using custom CUDA kernels
- Production ready logging, monitoring and tracing with Prometheus and Open Telemetry

(<https://huggingface.co/blog/falcon>; see also <https://github.com/huggingface/text-generation-inference/tree/main/router>; <https://www.anyscale.com/blog/continuous-batching-llm-inference>.)

1[F] generating, by the execution engine, a second set of output tokens by applying the transformer model to the second set of inputs for the second batch.	The Accused Functionality performs the step of generating, by the execution engine, a second set of output tokens by applying the transformer model to the second set of inputs for the second batch. The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 1[C]. (See https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L361-L362; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/sharded_client.rs#L98-L103; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/client.rs#L107-L108; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L61-L62; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L634-L646; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L602; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L648-L657; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L57-L59; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L93-L99; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L147; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L219; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L239; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L347; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L277-L278; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L277-L278;
--	--

	inference/blob/3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L140).
Claim 2. The method of claim 1, further comprising: responsive to determining that a request in the first batch of requests has been completed, providing output tokens generated for the completed request to a client device as a response to the request, and	The Accused Functionality performs the step of responsive to determining that a request in the first batch of requests has been completed, providing output tokens generated for the completed request to a client device as a response to the request. The Accused Functionality provides the output tokens generated for the completed request to a client device as a response to the request after determining that a request in the first batch of requests has been completed:

```

97      /// Generate one token for each request in the given batch
98      ///
99      /// Returns Generation for each request in batch
100     /// and the next cached batch
101     #[instrument(skip_all, fields(id = &batch.id, size = &batch.size))]
102     ✓ pub async fn prefill(
103         &mut self,
104         batch: Batch,
105     ) -> Result<(Vec<Generation>, Option<CachedBatch>)> {
106         let request = tonic::Request::new(PrefillRequest { batch: Some(batch) }).inject_context();
107         let response = self.stub.prefill(request).await?.into_inner();
108         Ok((response.generations, response.batch))
109     }
110
111     /// Generate one token for each request in the given cached batches
112     ///
113     /// Returns Generation for each request in batches
114     /// and the next cached batch
115     #[instrument(skip_all, fields(size = batches.iter().map(|batch| {batch.size}).sum:::<u32>()))]
116     ✓ pub async fn decode(
117         &mut self,
118         batches: Vec<CachedBatch>,
119     ) -> Result<(Vec<Generation>, Option<CachedBatch>)> {
120         let request = tonic::Request::new(DecodeRequest { batches }).inject_context();
121         let response = self.stub.decode(request).await?.into_inner();
122         Ok((response.generations, response.batch))
123     }
124 }

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/client.rs#L97-L124>).

```

155     /// Generate one token for each request in the given cached batches
156     ///
157     /// Returns Generation for each request in batches
158     /// and the next cached batch
159     #[instrument(skip_all, fields(size = batches.iter().map(| batch | {batch.size}).sum:::< u32 > ()))]
160     pub async fn decode(
161         &mut self,
162         batches: Vec<CachedBatch>,
163     ) -> Result<(Vec<Generation>, Option<CachedBatch>, DecodeTimings)> {
164         let futures: Vec<_> = self
165             .clients
166             .iter_mut()
167             .map(|client| Box::pin(client.decode(batches.clone())))
168             .collect();
169         #[allow(clippy::type_complexity)]
170         let results: Result<Vec<(Vec<Generation>, Option<CachedBatch>, DecodeTimings)>> =
171             join_all(futures).await.into_iter().collect();
172         let mut results = results?;
173
174         let (mut generations, next_batch, mut timings) =
175             results.pop().ok_or(ClientError::EmptyResults)?;

```

(https://github.com/huggingface/text-generation-inference/blob/main/router/client/src/sharded_client.rs).

The Accused Functionality, responsive to determining that a request in the first batch of requests has been completed checks completion of a request in the batch: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L789-L813.

The Accused Functionality propagates a “stopped” signal: <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L517-L529>.

The Accused Functionality provides output tokens generated for the completed request to a client device as a response to the request by first collecting output tokens: <https://github.com/huggingface/text->

	generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L164-L165; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L168-L174. The Accused Functionality then propagates output tokens: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/server.rs#L206. The Accused Functionality then sends a response to a client device: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/server.rs#L74-L76.
2[A] wherein the second batch of requests includes at least one of the remaining requests from the one or more requests and the new request.	The Accused Functionality performs the step wherein the second batch of requests includes at least one of the remaining requests from the one or more requests and the new request. The Accused Functionality creates a second batch of requests including at least one of the remaining requests from the one or more requests and the new request:

	<pre> 261 // Create a new span to link the batch back to this entry 262 let entry_batch_span = info_span!(parent: &entry.span, "infer"); 263 // Add relationships 264 next_batch_span.follows_from(&entry_batch_span); 265 entry_batch_span.follows_from(&next_batch_span); 266 // Update entry 267 entry.temp_span = Some(entry_batch_span); 268 269 batch_requests.push(Request { 270 id, 271 prefill_logprobs: entry.request.decoder_input_details, 272 inputs: entry.request.inputs.clone(), 273 truncate: entry.request.truncate, 274 parameters: Some(entry.request.parameters.clone()), 275 stopping_parameters: Some(entry.request.stopping_parameters.clone()), 276 top_n_tokens: entry.request.top_n_tokens, 277 }); </pre> (https://github.com/huggingface/text-generation-inference/blob/main/router/src/queue.rs). The Accused Functionality extends the first batch, including at least one of the remaining requests and the new request: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L318-L322. The Accused Functionality then runs the second batch: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L339-L341.
Claim 3. The method of claim 2, wherein the request is associated with a cache memory	The Accused Functionality performs the step wherein the request is associated with a cache memory in the execution engine dedicated for storing an internal state for the request, and responsive to determining

in the execution engine dedicated for storing an internal state for the request, and responsive to determining that the request has been completed, freeing the dedicated cache memory for the request in the execution engine.

that the request has been completed, freeing the dedicated cache memory for the request in the execution engine.

Defendant has proposed a construction of “dedicated cache memory”/“cache memory dedicated” that means “physical memory in, or that is a component part of, the execution engine.” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction.

The Accused Functionality associates a request with cache memory in the execution engine dedicated for storing an internal state for the request, and responsive to determining that the request has been completed, frees the dedicated cache memory for the request:

“Whenever either query is finished: either the model produce EOS (end of sentence) token or the query reached the allowed limit. We simply drop it from the batch, remove all the allocated memory and we can continue with the rest until nothing is left.”

(<https://github.com/huggingface/text-generation-inference/tree/main/router>)

```
415         // Generate one token for this new batch to have the attention past in cache
416         let new_cached_batch =
417             prefill(&mut client, new_batch, &mut new_entries, &generation_health)
418                 .instrument(span)
419                 .await;
420         // Reset waiting counter
421         waiting_tokens = 1;
422         // Extend current batch with the new batch
423         if let Some(new_cached_batch) = new_cached_batch {
424             entries.extend(new_entries);
425             batches.push(new_cached_batch);
426         }
```

(<https://github.com/huggingface/text-generation-inference/blob/main/router/src/infer.rs>).

```

558      // Retain only requests that are still in entries
559      batch.request_ids.retain(|id| entries.contains_key(id));
560
561      if batch.request_ids.is_empty() {
562          // All requests have been filtered out
563          // Next batch is now empty
564          // Clear it from the Python shards cache
565          // We unwrap here as we need to panic since we cannot recover if this method fails
566          client.clear_cache(Some(id)).await.unwrap();
567          None
568      } else {

```

(<https://github.com/huggingface/text-generation-inference/blob/main/router/src/infer.rs>).

The Accused Functionality clears the dedicated cache, responsive to determining that the request has been completed:

```

64      /// Clear the past generations cache
65      #[instrument(skip(self))]
66      pub async fn clear_cache(&mut self, batch_id: Option<u64>) -> Result<> {
67          let futures: Vec<_> = self
68              .clients
69              .iter_mut()
70              .map(|client| client.clear_cache(batch_id))
71              .collect();
72          join_all(futures).await.into_iter().collect()
73      }

```

(https://github.com/huggingface/text-generation-inference/blob/main/client/src/sharded_client.rs).

	The Accused Functionality allocates cache memory: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L349-L365. The Accused Functionality, responsive to determining that the request has been completed, frees the dedicated cache memory for the request in the execution engine: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L483-L485; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L434-L455; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L434-L455. The Accused Functionality then frees the dedicated cache memory for the request: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L306-L309; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L355-L358; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L367.
Claim 4. The method of claim 1, wherein the input for the at least one request is an output token from the first set of output tokens for the at least one request, and wherein a length of the sequence of input tokens for the new request is different from a length of the	The Accused Functionality performs the step of wherein the input for the at least one request is an output token from the first set of output tokens for the at least one request, and wherein a length of the sequence of input tokens for the new request is different from a length of the output token for the at least one request. Defendant has proposed a construction of “a length of the sequence of input tokens for the new request is different from a length of the output token for the at least one request” where “‘length’ refers to a ‘number or quantity of discrete <units>’ for a given context— e.g., the number of “tokens” in a sequence, such as “a [number or quantity of discrete tokens] of the sequence of input tokens for the new request is different from a [number or quantity of discrete tokens] of an input for at least one request.” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction.

output token for the at least one request.

The Accused Functionality uses the output tokens generated from at least one request as input for a new sequence of input tokens, where the length of the sequence of input tokens is different from the length of the output token:

```
415          // Generate one token for this new batch to have the attention past in cache
416          let new_cached_batch =
417              prefill(&mut client, new_batch, &mut new_entries, &generation_health)
418                  .instrument(span)
419                  .await;
420          // Reset waiting counter
421          waiting_tokens = 1;
422          // Extend current batch with the new batch
423          if let Some(new_cached_batch) = new_cached_batch {
424              entries.extend(new_entries);
425              batches.push(new_cached_batch);
426          }
```

(<https://github.com/huggingface/text-generation-inference/blob/main/router/src/infer.rs>).

```
309          // Final batch size
310          let size = batch_requests.len() as u32;
311          next_batch_span.record("batch_size", size);
312
313          let batch = Batch {
314              id: self.next_batch_id,
315              requests: batch_requests,
316              size,
317              max_tokens: (prefill_tokens + decode_tokens),
318          };
```

	(https://github.com/huggingface/text-generation-inference/blob/main/router/src/queue.rs). The Accused Functionality retrieves an output token from an iteration: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L655-L657; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L732; The Accused Functionality then uses the saved output token as the input: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L634-L646.
Claim 5. The method of claim 4, wherein the execution engine includes a cache memory for maintaining a key cache tensor for storing keys and a value cache tensor for storing values for the at least one request, and	The Accused Functionality performs the step wherein the execution engine includes a cache memory for maintaining a key cache tensor for storing keys and a value cache tensor for storing values for the at least one request. The Accused Functionality includes cache memory for maintaining a key cache tensor for storing keys and a value cache tensor for storing values for the at least one request. “In the decoding part of generation, all the attention keys and values generated for previous tokens are stored in GPU memory for reuse. This is called KV cache, and it may take up a large amount of memory for large models and long sequences.” (https://huggingface.co/docs/text-generation-inference/en/conceptual/paged_attention).

	<pre> 415 // Generate one token for this new batch to have the attention past in cache 416 let new_cached_batch = 417 prefill(&mut client, new_batch, &mut new_entries, &generation_health) 418 .instrument(span) 419 .await; 420 // Reset waiting counter 421 waiting_tokens = 1; 422 // Extend current batch with the new batch 423 if let Some(new_cached_batch) = new_cached_batch { 424 entries.extend(new_entries); 425 batches.push(new_cached_batch); 426 } </pre> (https://github.com/huggingface/text-generation-inference/blob/main/router/src/infer.rs). The Accused Functionality provides for key/value cache tensor maintenance: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flan_causal_lm.py#L367.
5[B] wherein after scheduling the second batch of requests, allocating, by the execution engine, a new cache memory dedicated to maintaining a key cache tensor and a value cache tensor for the new request.	The Accused Functionality performs the step wherein after scheduling the second batch of requests, allocating, by the execution engine, a new cache memory dedicated to maintaining a key cache tensor and a value cache tensor for the new request. Defendant has proposed a construction of “dedicated cache memory”/“cache memory dedicated” that means “physical memory in, or that is a component part of, the execution engine.” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction. The Accused Functionality allocates a new cache memory dedicated to maintaining a key cache tensor and value cache tensor for the new request, after scheduling the second batch of requests:

	<pre> 415 // Generate one token for this new batch to have the attention past in cache 416 let new_cached_batch = 417 prefill(&mut client, new_batch, &mut new_entries, &generation_health) 418 .instrument(span) 419 .await; 420 // Reset waiting counter 421 waiting_tokens = 1; 422 // Extend current batch with the new batch 423 if let Some(new_cached_batch) = new_cached_batch { 424 entries.extend(new_entries); 425 batches.push(new_cached_batch); 426 } </pre> (https://github.com/huggingface/text-generation-inference/blob/main/router/src/infer.rs). The Accused Functionality allocates cache memory: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L349-L365. The Accused Functionality also maintains a reference to the allocated cache memory: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L860.
Claim 6. The method of claim 5, wherein after generating the second set of output tokens, a length of the key cache tensor for the at least one request is different from a length of a key cache tensor	The Accused Functionality performs the step wherein after generating the second set of output tokens, a length of the key cache tensor for the at least one request is different from a length of a key cache tensor for the new request, and a length of the value cache tensor for the at least one request is different from a length of a value cache tensor for the new request. Defendant has proposed a construction of “a length of the key cache tensor for the at least one request is different from a length of a key cache tensor for the new request, and a length of the value cache tensor for the at least one request is different from a length of a value cache tensor for the new request” where “‘length’ refers to a ‘number or quantity of discrete <units>’ for a given context— e.g., the number of

for the new request, and a length of the value cache tensor for the at least one request is different from a length of a value cache tensor for the new request.

“tokens” in a sequence, such as “a [number or quantity of discrete tokens] of the sequence of input tokens for the new request is different from a [number or quantity of discrete tokens] of an input for at least one request.” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction.

The Accused Functionality allows the length of the key cache tensor and value cache tensor for the at least one request to be different from a length of the cache tensor for the new request:

```
1016             # Shard generations
1017             # All generations will be appended in the rust sharded client
1018             if i % self.world_size == self.rank:
1019                 if stop:
1020                     # Decode generated tokens
1021                     output_text, _, _ = self.decode_token(
1022                         all_input_ids,
1023                         prefix_offset=len(all_input_ids)
1024                         - stopping_criteria.current_tokens
1025                         - 1,
1026                         read_offset=len(all_input_ids)
1027                         - stopping_criteria.current_tokens,
1028                         skip_special_tokens=True,
1029                     )
1030                     generated_text = GeneratedText(
1031                         output_text,
1032                         stopping_criteria.current_tokens,
1033                         reason,
1034                         seed if do_sample else None,
1035                     )
```

(https://github.com/huggingface/text-generation-inference/blob/main/server/text_generation_server/models/flash_causal_lm.py).

```

1104         # Update values
1105         batch.input_lengths[i] = input_length + n_accepted_ids
1106         if batch.input_lengths[i] > batch.max_seqlen:
1107             batch.max_seqlen = batch.input_lengths[i]
1108         batch.prefix_offsets[i] = prefix_offset
1109         batch.read_offsets[i] = read_offset
1110         batch.all_input_ids[i] = all_input_ids

```

(https://github.com/huggingface/text-generation-inference/blob/main/server/text_generation_server/models/flash_causal_lm.py).

The Accused Functionality generates the token() function: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L617.

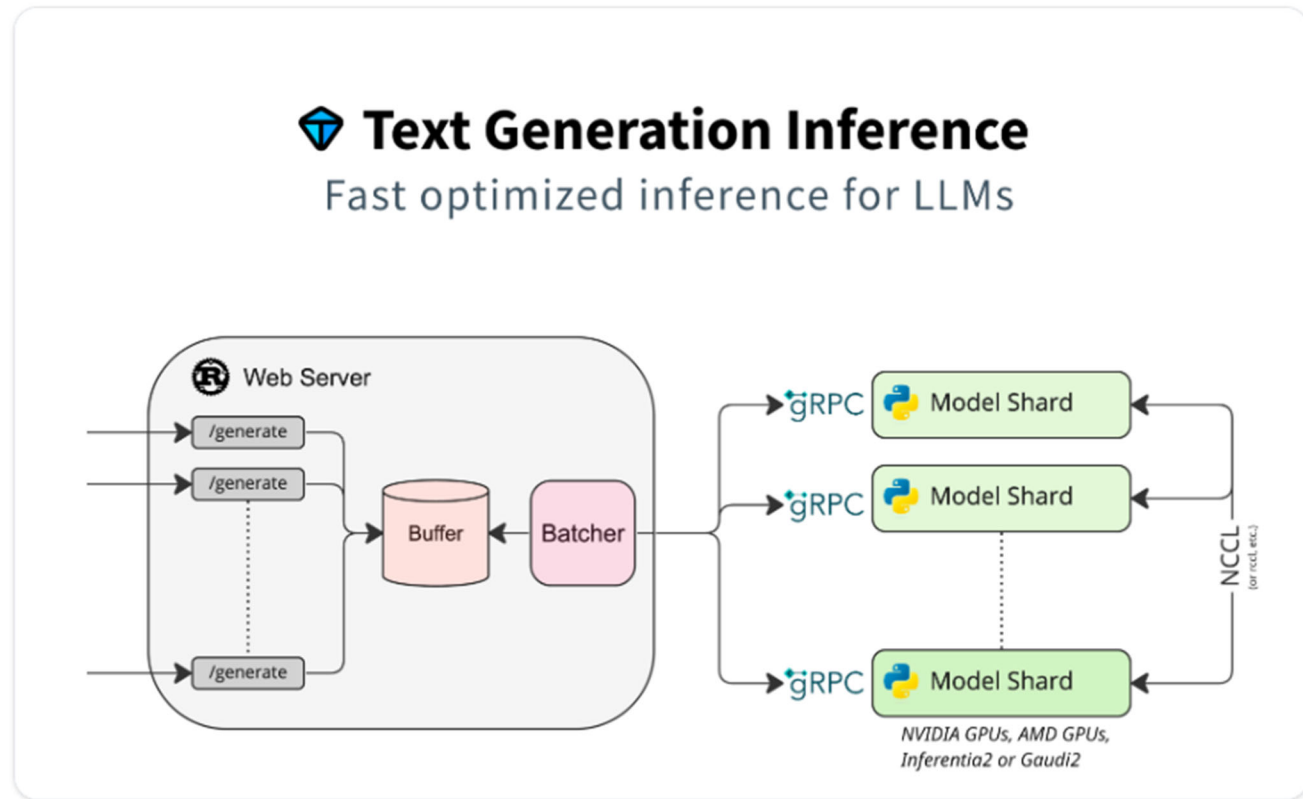
The Accused Functionality then executes the generate_token() function using prefill() and decode(): <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L313> -> https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L61; <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L339> -> https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L88.

The Accused Functionality increments input_lengths by 1 after every execution of the generate_token() function: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L848-L851.

The Accused Functionality then initializes the input_lengths: https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L851.

	inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L57-L59 -> https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L140-L142 .
Claim 7. The method of claim 1, after receiving the new request from the client device, determining, by the scheduler, that there is insufficient memory available to execute the second batch of requests on a second execution engine different from the execution engine, and responsive to the determination for the second execution engine, determining whether the execution engine has the memory available to execute the second batch of requests.	The Accused Functionality performs the step of after receiving the new request from the client device, determining, by the scheduler, that there is insufficient memory available to execute the second batch of requests on a second execution engine different from the execution engine, and responsive to the determination for the second execution engine, determining whether the execution engine has the memory available to execute the second batch of requests. The Accused Functionality schedules the second batch of requests responsive to determining that the execution engine has memory available to execute the second batch of requests: <pre> 287 let token_budget = max_batch_total_tokens - batch_max_tokens; 288 289 // Try to get a new batch 290 if let Some((mut new_entries, new_batch, span)) = 291 queue.next_batch(min_size, token_budget).await </pre> (https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L287-L291). <pre> 187 if (prefill_tokens + decode_tokens) > token_budget { 188 // Entry is over budget 189 // Add it back to the front 190 self.entries.push_front((id, entry)); 191 break; 192 } </pre> (https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/queue.rs#L187-L192).

The Accused Functionality uses multiple GPUs for execution and can determine which execution engine to use:



Text Generation Inference implements many optimizations and features, such as:

- Simple launcher to serve most popular LLMs
- Production ready (distributed tracing with Open Telemetry, Prometheus metrics)
- Tensor Parallelism for faster inference on multiple GPUs

	(https://huggingface.co/docs/text-generation-inference/en/index).
Claim 8. The method of claim 1, wherein the execution engine is configured as a graphics processing unit (GPU) or a tensor processing unit (TPU).	The Accused Functionality performs the step wherein the execution engine is configured as a graphics processing unit (GPU) or a tensor processing unit (TPU). <pre> 311 # Create on CPU to only move to GPU once instead of at every copy 312 start_seq = torch.empty(len(request_ids), dtype=torch.int32) 313 end_seq = torch.empty(len(request_ids), dtype=torch.int32) 314 start_seq_q = self.start_seq_q[: len(request_ids)] 315 end_seq_q = self.end_seq_q[: len(request_ids)] 316 max_seqlen = 0 </pre> (https://github.com/huggingface/text-generation-inference/blob/main/server/text_generation_server/models/flash_causal_lm.py). <pre> 888 # We do two for loops as the first one can run completely asynchronously from the GPU while for the second 889 # one, we need to first do a GPU <-> CPU sync 890 # It is faster if we delay this sync for the maximum amount of time </pre> (https://github.com/huggingface/text-generation-inference/blob/main/server/text_generation_server/models/flash_causal_lm.py). <pre> 434 # Move to GPU now that we have the whole tensor 435 slot_indices = slot_indices.to(device) </pre> (https://github.com/huggingface/text-generation-inference/blob/main/server/text_generation_server/models/flash_causal_lm.py).
Claim 9. The method of claim 1, wherein each token in the sequence of input	The Accused Functionality performs the step wherein each token in the sequence of input tokens represents a text unit. The Accused Functionality tokenizes the submitted text into a sequence of input tokens:

tokens represents a text unit.

```
73  ✓ pub async fn tokenize(  
74      &self,  
75      inputs: String,  
76      truncate: Option<usize>,  
77  ) -> Result<Option<(tokenizers::Encoding, String)>, ValidationError> {  
78      // If we have a fast tokenizer  
79      if let Some(sender) = &self.sender {  
80          // Create response channel  
81          let (response_sender, response_receiver) = oneshot::channel();  
82          // Send request to the background validation task  
83          // Unwrap is safe here  
84          sender  
85              .send(((inputs, truncate), response_sender, Span::current()))  
86              .unwrap();  
87  
88          // Await on response channel  
89          // Unwrap is safe here  
90          let encoding = response_receiver.await.unwrap()?;  
91          Ok(Some(encoding))  
92      } else {  
93          Ok(None)  
94      }  
95  }
```

(<https://github.com/huggingface/text-generation-inference/blob/main/router/src/validation.rs>).

Claim 10. A non-transitory computer-readable storage medium storing computer program instructions executable to perform operations for dynamically executing batches of requests on one or more execution engines running a machine-learning transformer model, the operations comprising:	Defendant has designed and developed, and has used, sold, offered for sale, and/or otherwise made available (and continues to use, sell, offer for sale, and/or otherwise make available), the Accused Products and Services that use Text Generation Inference (“TGI” or the “Accused Functionality”). The Accused Functionality performs operations for dynamically executing batches of requests on one or more execution engines running a machine-learning transformer model. <i>See</i> https://huggingface.co/text-generation-inference.² The Accused Products include Text Generation Inference (TGI), and any product, service, platform, solutions, model, dataset, application, or toolkit that Hugging Face has made, sold, offered for sale, advertised, marketed, used, or imported into the United States that incorporates, applies, or otherwise uses Text Generation Inference (TGI), including but not limited to the TGI toolkit distributed or otherwise made available by Hugging Face (including open source and under license), Spaces, Inference Endpoints, Enterprise Hub (formerly known as Private Hub), Hugging Face Hub, HuggingChat, OpenAssistant, and Docker Hub containers, including all past, current, and future versions of these products and services, even if they are not explicitly referred to by these product names, including any that may be released during the pendency of this matter (“Accused Products”). FriendliAI identifies the above Accused Products, but further discovery may be necessary to reveal additional Accused Products. FriendliAI reserves the right to modify, supplement, amend, or otherwise alter these disclosures based on forthcoming discovery. Hugging Face indirectly infringes the Asserted Patents through contributory infringement and induced infringement. Hugging Face had knowledge of the Asserted Patents (<i>see</i> Amended Compl., at, e.g., ¶¶ 43, 50, 53, 63, 66). Hugging Face contributorily infringes each of the Asserted Patents by selling and/or distributing a component especially designed for use in the patented invention. Specifically, Hugging Face sells and/or distributes TGI. Further, TGI is (at a minimum) a material component of the claimed invention, and is not suitable for substantial non-infringing uses. The batching efficiencies that Hugging Face touts in its description of TGI are at the core of many of the Hugging Face products at issue in this case. Hugging Face induces infringement of the Asserted Patents by actively and knowingly aiding and abetting Hugging Face users’ and business partners’ direct infringement. Hugging Face had knowledge of the infringement because it continued to use and promote technology infringing the Asserted Patents since at least July 20, 2023, when it was placed on notice that the Accused Products infringed the claims of the asserted patents. FriendliAI further incorporates by reference the allegations in the Amended Complaint,
---	---

² By charting the preamble, Plaintiff does not take any position as to whether the preamble is a limitation of the claim.

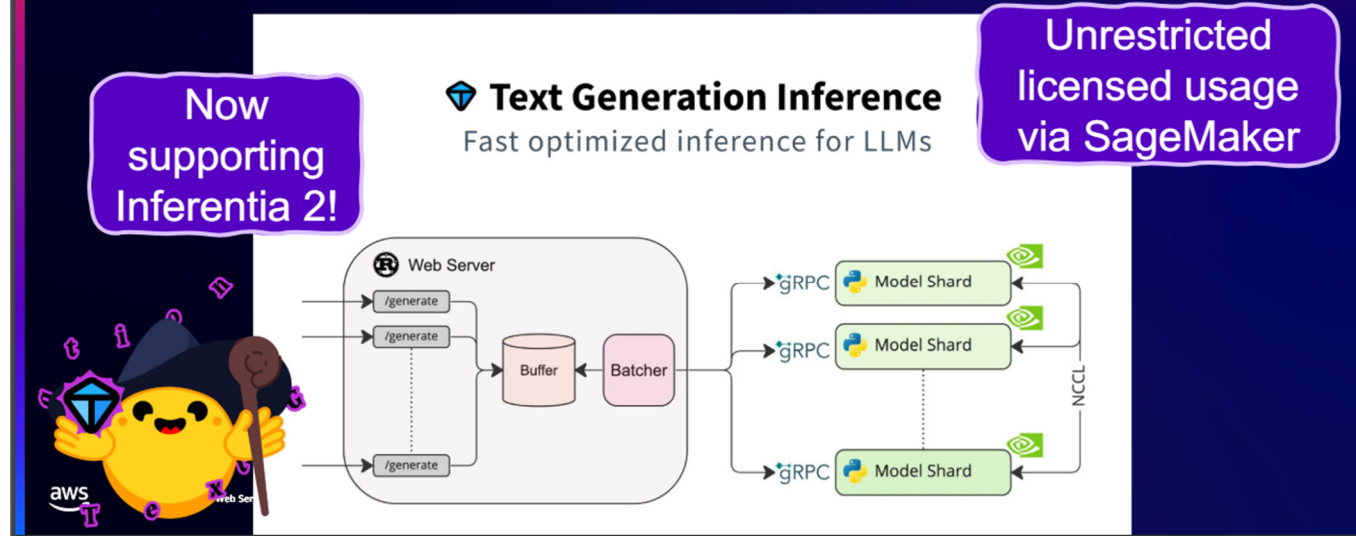
at, e.g., ¶¶ 41-42. Hugging Face provides users and business partners with instructions on how to use TGI in an infringing manner. Hugging Face provides tools and instructions on its website on how to use TGI to perform steps that result in the directly infringing use. Hugging Face also advertises or promotes the use of TGI. Additionally, Hugging Face is responsible for and exercises control over the manufacturing and development of TGI. Hugging Face has actively encouraged its customers to commit direct infringement of the asserted claims with knowledge that the acts encouraged constitute infringement and a specific intent to infringe the asserted claims.

The Accused Products also indirectly infringe at least because Hugging Face provides TGI to customers to execute on their own servers, Hugging Face induces customers to infringe the '775 patent claims by providing TGI and instructions to execute TGI on third-party systems, or Hugging Face indirectly infringes by providing TGI in any other fashion to be used with other software or hardware. (*See* <https://github.com/huggingface/text-generation-inference>).

Hugging Face also indirectly infringes when it provides enterprise consulting and support to customers to implement TGI on either the customer's own hardware or in their own particular environment. (*See* DEF_000298; FRIENDLIAI_00001163). In some cases, this includes Hugging Face providing custom TGI containers optimized to execute on the customer's hardware or in the customer's environment. *Id.* Hugging Face also indirectly infringes when it licenses or otherwise provides TGI to partners and collaborators to execute in their environment, and encourages users to use TGI in their partners' environment. (*See* DEF_360181; DEF_360159; FRIENDLIAI_00001163).

Cost effective: With Text Generation Inference

TGI is designed for maximum throughput for LLM inference



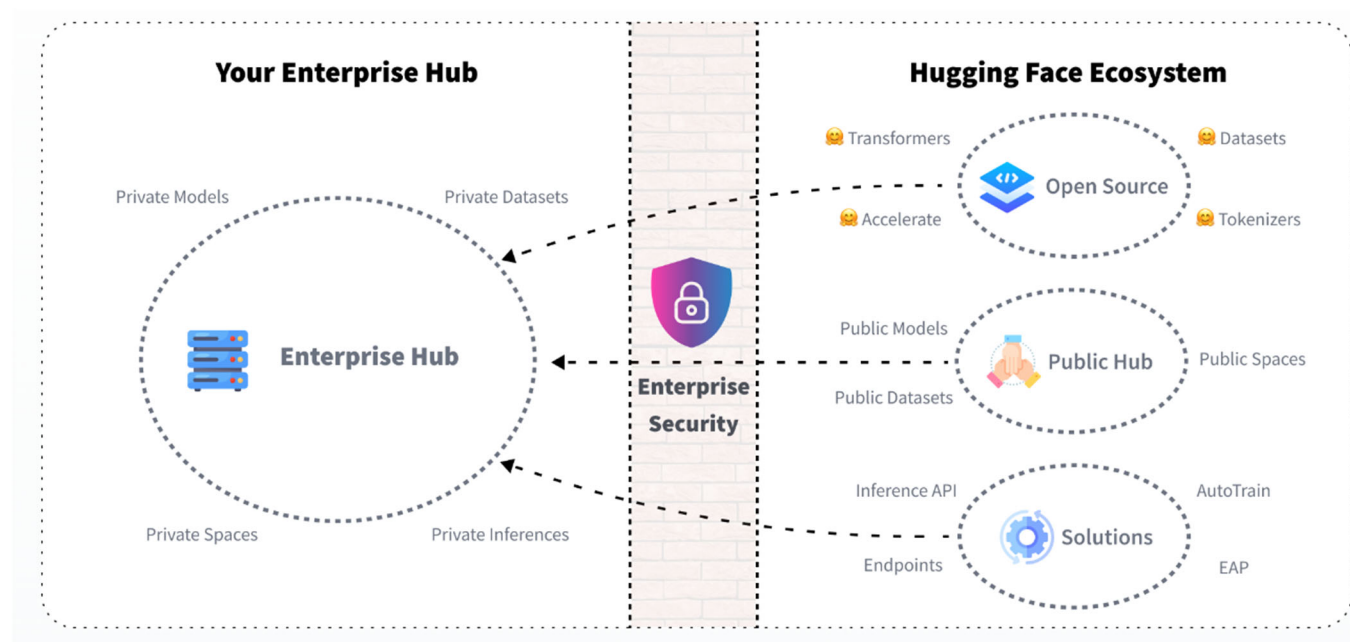
(See <https://www.youtube.com/watch?v=cTyXDhiltXw>; see also https://d1.awsstatic.com/events/Summits/reinvent2023/AIM328_Accelerate-FM-development-with-Amazon-SageMaker-JumpStart.pdf).

Hugging Face's Spaces product infringes the claims of the '775 patent by incorporating TGI into its execution environment: DEF_347045; DEF_347048; DEF_347057; DEF_347071; https://huggingface.co/docs/text-generation-inference/en/basic_tutorials/consuming_tgi ("ChatUI can automatically consume the TGI server and even provides an option to switch between different TGI endpoints. You can try it out at Hugging Chat, or use the ChatUI Docker Space to deploy your own Hugging Chat to Spaces."); https://huggingface.co/spaces/ysharma/Explore_llamav2_with_TGI. Customers are encouraged to create their own applications on Spaces (e.g., Chatbots) using TGI. (See

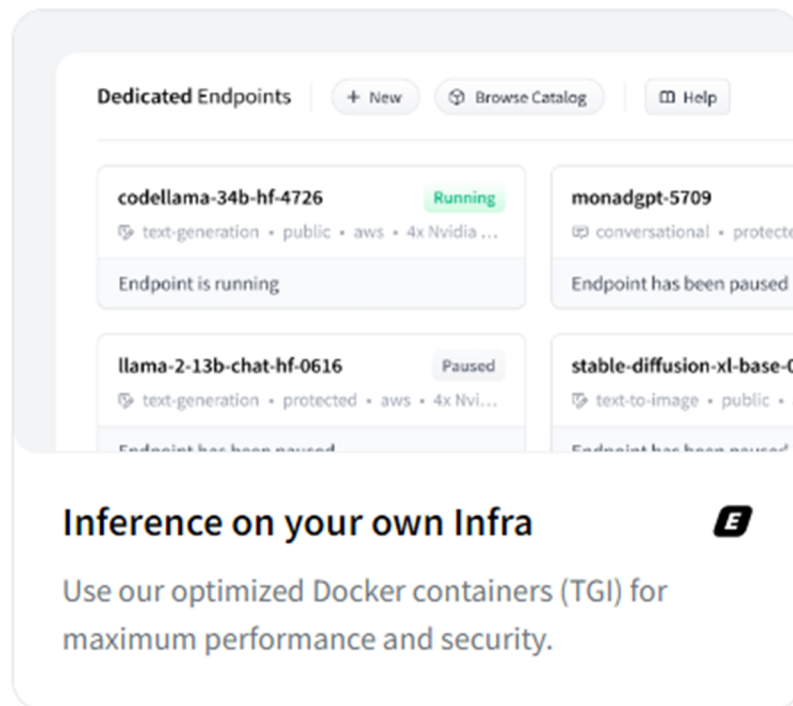
<https://github.com/huggingface/chat-ui>; <https://huggingface.co/chat/>; DEF_000050;
https://huggingface.co/docs/text-generation-inference/en/basic_tutorials/consuming_tgi).

Hugging Face's Inference Endpoints product infringes the claims of the '775 patent by incorporating TGI into its execution environment: <https://huggingface.co/blog/inference-endpoints-llm>; <https://huggingface.co/text-generation-inference>. Hugging Face provides a custom TGI container to users of Inference Endpoints that allows users to easily deploy TGI onto various hardware systems. Hugging Face builds and deploys TGI for users of Inference Endpoints. (DEF_347031; DEF_347040; DEF_347045; DEF_347074; DEF_347080; DEF_347083; DEF_347085; DEF_347112-DEF_347116; DEF_347040).

Hugging Face's Enterprise Hub product infringes the claims of the '775 patent by incorporating TGI into its execution environment:



<https://huggingface.co/enterprise>; see <https://huggingface.co/text-generation-inference>.



(<https://huggingface.co/enterprise>; see also FRIENDLIAI_00001163. Enterprise Hub implements TGI for its customers, as evident from its source code. (DEF_30711-DEF_30722; DEF_30726; DEF_30728; DEF_30730- DEF_30732; DEF_30787; DEF_30797; DEF_30800). The Hugging Face Hub infringes for at least the same reasons as the Enterprise Hub, as the source code overlaps for the two products.

Hugging Face’s HuggingChat product infringes the claims of the ’775 patent by incorporating TGI into its execution environment: <https://huggingface.co/text-generation-inference>; <https://huggingface.co/chat/>; <https://huggingface.co/docs/text-generation-inference/index> (“Text Generation Inference is used in production by multiple projects, such as: Hugging Chat, an open-source interface for open-access models, such as Open Assistant and Llama.”).

	Hugging Face deploys the OpenAssistant product that infringes the claims of the '775 patent by incorporating TGI into its execution environment: https://huggingface.co/text-generation-inference (“Text Generation Inference is already used by customers such as IBM, Grammarly, and the Open-Assistant initiative implements optimization for all supported model architectures”); https://huggingface.co/chat/; https://huggingface.co/docs/text-generation-inference/index (“Text Generation Inference is used in production by multiple projects, such as: Hugging Chat, an open-source interface for open-access models, such as Open Assistant and Llama. OpenAssistant, an open-source community effort to train LLMs in the open”). Hugging Face’s Docker Hub product infringes the claims of the '775 patent by incorporating TGI into its execution environment and by distributing TGI as a Docker Container: https://huggingface.co/docs/text-generation-inference/quicktour (“Quick Tour - The easiest way of getting started is using the official Docker container.”); https://huggingface.co/docs/text-generation-inference/installation (“This section explains how to install the CLI tool as well as installing TGI from source. The strongly recommended approach is to use Docker, as it does not require much setup. Check the Quick Tour to learn how to run TGI with Docker.”); https://github.com/huggingface/text-generation-inference.
10[A] receiving, by a serving system, one or more requests for execution, the serving system including a scheduler and one or more execution engines each coupled to access a machine-learning transformer model including at least a set of decoders;	The Accused Functionality performs the step of receiving, by a serving system, one or more requests for execution, the serving system including a scheduler and one or more execution engines each coupled to access a machine-learning transformer model including at least a set of decoders. The Accused Functionality performs the step of receiving, by a serving system, one or more requests for execution. The Accused Functionality receives a request for execution:

```

33 /// Generate tokens if 'stream == false' or a stream of token if 'stream == true'
34 #![utoipa::path(
35     post,
36     tag = "Text Generation Inference",
37     path = "/",
38     request_body = CompatGenerateRequest,
39     responses(
40         (status = 200, description = "Generated Text",
41             content(
42                 ("application/json" = GenerateResponse),
43                 ("text/event-stream" = StreamResponse),
44             )),
45         (status = 424, description = "Generation Error", body = ErrorResponse,
46             example = json ! ({"error": "Request failed during generation"})),
47         (status = 429, description = "Model is overloaded", body = ErrorResponse,
48             example = json ! ({"error": "Model is overloaded"})),
49         (status = 422, description = "Input validation error", body = ErrorResponse,
50             example = json ! ({"error": "Input validation error"})),
51         (status = 500, description = "Incomplete generation", body = ErrorResponse,
52             example = json ! ({"error": "Incomplete generation"})),
53     )
54 )]
55 #![instrument(skip(infer, req))]
56 ✓ async fn compat_generate(
57     default_return_full_text: Extension<bool>,
58     infer: Extension<Infer>,
59     req: Json<CompatGenerateRequest>,
60 ) -> Result<Response, (StatusCode, Json<ErrorResponse>)> {
61     let mut req = req.0;
62
63     // default return_full_text given the pipeline_tag
64     if req.parameters.return_full_text.is_none() {
65         req.parameters.return_full_text = Some(default_return_full_text.0)
66     }
67
68     // switch on stream
69     if req.stream {
70         Ok(generate_stream(infer, Json(req.into())))
71             .await
72             .into_response()
73     } else {
74         let (headers, generation) = generate(infer, Json(req.into())).await?;
75         // wrap generation inside a Vec to match api-inference
76         Ok((headers, Json(vec![generation.0])).into_response())
77     }
78 }

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/server.rs#L33-L78>).

The Accused Functionality includes a scheduler:

```

239 /// Batching logic
240 /// Will be launched in a background Tokio task
241 ///
242 /// Batches requests and sends them to the inference server
243 ✓ async fn batching_task(
244     mut client: ShardedClient,
245     waiting_served_ratio: f32,
246     max_batch_total_tokens: u32,
247     max_waiting_tokens: usize,
248     queue: Queue,
249     shared: Arc<Shared>,
250     generation_health: Arc<AtomicBool>,
251 ) {
252     // Infinite loop
253     loop {
254         // Wait for a notification from the Infer struct
255         shared.batching_task.notified().await;
256
257         // Get the next batch from the queue
258         // This batch might be smaller than the maximum batch size if there are not enough requests
259         // waiting in the queue
260         while let Some((mut entries, batch, span)) =
261             queue.next_batch(None, max_batch_total_tokens).await
262         {

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L239-L348>; see also <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/server.rs#L580-L589>)

The Accused Functionality scheduler puts the received requests into a queue:

```

115         // Append the request to the queue
116         self.queue.append(Entry {
117             request: valid_request,
118             response_tx,
119             span: Span::current(),
120             temp_span: None,
121             queue_time: Instant::now(),
122             batch_time: None,
123         });
124
125         // Notify the background task that we have a new entry in the queue that needs
126         // to be batched
127         self.shared.batching_task.notify_one();

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L115-L127>; see also <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L239-L348>)

The Accused Functionality includes one or more one or more execution engines, comprising one or more shards:

```
987     spawn_shards(  
988         num_shard,  
989         &args,  
990         shutdown.clone(),  
991         &shutdown_receiver,  
992         shutdown_sender,  
993         &status_receiver,  
994         status_sender,  
995         running.clone(),  
996     );  
---
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/launcher/src/main.rs#L987-L996>; see also https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L97-L155; see also https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/cli.py#L154-L155).

Each of the execution engines in the Accused Functionality is coupled to access a machine-learning transformer model including at least a set of decoders. The Accused Functionality gets a model:

```
123     try:  
124         model = get_model(model_id, revision, sharded, quantize, trust_remote_code)  
125     except Exception:  
126         logger.exception("Error when initializing model")  
127         raise
```

(https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L123-L124).

The machine-learning transformer models used by the Accused Functionality include at least a set of decoders:

	<pre> 155 if model_type == "bloom": 156 return BLOOMSharded(157 model_id, revision, quantize=quantize, trust_remote_code=trust_remote_code 158) 159 160 elif model_type == "gpt_neox": 161 if FLASH_ATTENTION: 162 return FlashNeoXSharded(163 model_id, 164 revision, 165 quantize=quantize, 166 trust_remote_code=trust_remote_code, 167) 168 elif sharded: 169 return GPTNeoXSharded(170 model_id, 171 revision, 172 quantize=quantize, 173 trust_remote_code=trust_remote_code, 174) 175 else: 176 return CausalLM(177 model_id, 178 revision, 179 quantize=quantize, 180 trust_remote_code=trust_remote_code, 181) 182 183 elif model_type == "llama": 184 if FLASH_ATTENTION: 185 return FlashLlama(186 model_id, 187 revision, 188 quantize=quantize, 189 trust_remote_code=trust_remote_code, 190) </pre> (https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/_init_.py#L98-L276).
10[B] scheduling, by the scheduler, a batch of requests including	The Accused Functionality performs the step of scheduling, by the scheduler, a batch of requests including the one or more requests for execution on an execution engine.

the one or more requests for execution on an execution engine;

The Accused Functionality performs the scheduling of a batch of requests, including the one or more requests for execution on an execution engine. The Accused Functionality removes a batch of requests, comprising the one or more requests, from the scheduling queue, then schedules execution of the first iteration on the execution engine:

```
252         // Infinite loop
253         loop {
254             // Wait for a notification from the Infer struct
255             shared.batching_task.notified().await;
256
257             // Get the next batch from the queue
258             // This batch might be smaller than the maximum batch size if there are not enough requests
259             // waiting in the queue
260             while let Some((mut entries, batch, span)) =
261                 queue.next_batch(None, max_batch_total_tokens).await
262             {
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L260-L261>).

```
263             let mut cached_batch = prefill(&mut client, batch, &mut entries, &generation_health)
264                 .instrument(span)
265                 .await;
266             let mut waiting_tokens = 1;
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L263-L265>).

The Accused Functionality also schedules execution of further iterations on the execution engine:

```

339         cached_batch = decode(&mut client, batches, &mut entries, &generation_health)
340         .instrument(next_batch_span)
341         .await;
342         waiting_tokens += 1;
343     }

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L339-L341>).

10[C] generating, by the execution engine, a first set of output	The Accused Functionality performs the step of generating, by the execution engine, a first set of output tokens by applying the transformer model to a first set of inputs for the batch of requests, wherein
--	--

tokens by applying the transformer model to a first set of inputs for the batch of requests, wherein applying the transformer model comprises applying at least one batch operation to one or more input tensors associated with the batch of requests;

applying the transformer model comprises applying at least one batch operation to one or more input tensors associated with the batch of requests.

The Accused Functionality first applies the transformer model to a first set of inputs for the batch of requests, wherein applying the transformer model comprises applying at least one batch operation to one or more input tensors associated with the batch of request:

```
361         match client.prefill(batch).await {  
362             Ok((generations, next_batch)) => {
```

- (<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L361-L362>; see also https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L147; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L219; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L239; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L347; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L277-L278).

The Accused Functionality then generates tokens from applying the transformer model to the set of inputs in the batch:

```
90      /// Generate one token for each request in the given batch
91      ///
92      /// Returns Generation for each request in batch
93      /// and the next cached batch
94      #[instrument(skip_all, fields(id = &batch.id, size = &batch.size))]
95 ✓ pub async fn prefill(
96     &mut self,
97     batch: Batch,
98 ) -> Result<(Vec<Generation>, Option<CachedBatch>)> {
99     let futures: Vec<_> = self
100         .clients
101         .iter_mut()
102         .map(|client| Box::pin(client.prefill(batch.clone())))
103         .collect();
```

(https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/sharded_client.rs#L98-L103).

```
97      /// Generate one token for each request in the given batch
98      ///
99      /// Returns Generation for each request in batch
100      /// and the next cached batch
101      #[instrument(skip_all, fields(id = &batch.id, size = &batch.size))]
102 ✓ pub async fn prefill(
103     &mut self,
104     batch: Batch,
105 ) -> Result<(Vec<Generation>, Option<CachedBatch>)> {
106     let request = tonic::Request::new(PrefillRequest { batch: Some(batch) }).inject_context();
107     let response = self.stub.prefill(request).await?.into_inner();
108     Ok((response.generations, response.batch))
109 }
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/client.rs#L107-L108>).

The generated tokens are passed back to the Accused Functionality server, and another batch can be scheduled to execute:

```
61     generations, next_batch = self.model.generate_token(batch)
62     self.cache.set(next_batch)
```

(https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L61-L62).

In order to perform these steps, TGI creates the first batch and adds the input tokens to the batch. (See https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L57-L59; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L93-L99; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L147; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L219; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L239).

The Accused Functionality performs at least one batch operation on the input tensors. (See https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L347; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L277-L278; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L140).

10[D] receiving, by a request processor, a new request from a client device, the new request including a sequence of input tokens;	The Accused Functionality performs the step of receiving, by a request processor, a new request from a client device, the new request including a sequence of input tokens. Defendant has proposed a construction of “a new request” that means “a request different from each of the one or more requests of the [first] batch of requests.” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction. The Accused Functionality receives, by a request processor, a new request from a client device, the new request including a sequence of input token:

```

33 /// Generate tokens if `stream == false` or a stream of token if `stream == true`
34 #[utoipa::path(
35     post,
36     tag = "Text Generation Inference",
37     path = "/",
38     request_body = CompatGenerateRequest,
39     responses(
40         (status = 200, description = "Generated Text",
41             content(
42                 ("application/json" = GenerateResponse),
43                 ("text/event-stream" = StreamResponse),
44             )),
45         (status = 424, description = "Generation Error", body = ErrorResponse,
46             example = json ! ({"error": "Request failed during generation"})),
47         (status = 429, description = "Model is overloaded", body = ErrorResponse,
48             example = json ! ({"error": "Model is overloaded"})),
49         (status = 422, description = "Input validation error", body = ErrorResponse,
50             example = json ! ({"error": "Input validation error"})),
51         (status = 500, description = "Incomplete generation", body = ErrorResponse,
52             example = json ! ({"error": "Incomplete generation"})),
53     )
54 )]
55 #[instrument(skip(infer, req))]
56 ✓ async fn compat_generate(
57     default_return_full_text: Extension<bool>,
58     infer: Extension<Infer>,
59     req: Json<CompatGenerateRequest>,
60 ) -> Result<Response, (StatusCode, Json<ErrorResponse>)> {
61     let mut req = req.0;
62
63     // default return_full_text given the pipeline_tag
64     if req.parameters.return_full_text.is_none() {
65         req.parameters.return_full_text = Some(default_return_full_text.0)
66     }
67
68     // switch on stream
69     if req.stream {
70         Ok(generate_stream(infer, Json(req.into())))
71             .await
72             .into_response()
73     } else {
74         let (headers, generation) = generate(infer, Json(req.into())).await?;
75         // wrap generation inside a Vec to match api-inference
76         Ok((headers, Json(vec![generation.0])).into_response())
77     }
78 }

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/server.rs#L33-L78>).

The Accused Functionality then tokenizes the input text into a sequence of input tokens:

```

97     batch_tokenized_inputs = tokenizer(
98         batch_inputs, truncation=True, max_length=max_truncation
99     )["input_ids"]

```

	(https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flan_causal_lm.py#L97-L99).
10[E] scheduling, by the scheduler, a second batch of requests additionally including the new request for execution on the execution engine, the	The Accused Functionality performs the step of scheduling, by the scheduler, a second batch of requests additionally including the new request for execution on the execution engine, the second batch of requests scheduled responsive to determining that the execution engine has memory available to execute the second batch of requests, wherein in a second set of inputs for the second batch of requests, a length of the sequence of input tokens for the new request is different from a length of an input for at least one request other than the new request.

second batch of requests scheduled responsive to determining that the execution engine has memory available to execute the second batch of requests, wherein in a second set of inputs for the second batch of requests, a length of the sequence of input tokens for the new request is different from a length of an input for at least one request other than the new request; and

Defendant has proposed a construction of “second batch of requests” that means “a batch that includes at least one request from the [first] batch of requests.” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction.

Defendant has proposed a construction of “a length of the sequence of input tokens for the new request is different from a length of an input for [the] at least one request [other than the new request]” where “‘length’ refers to a ‘number or quantity of discrete <units>’ for a given context— e.g., the number of ‘tokens’ in a sequence, such as ‘a [number or quantity of discrete tokens] of the sequence of input tokens for the new request is different from a [number or quantity of discrete tokens] of an input for at least one request.’” Plaintiff disagrees with this construction, or the need to construe this term. Regardless, the accused functionality infringes under the plain meaning or under Defendant’s proposed construction.

The Accused Functionality performs the step of scheduling, by the scheduler, a second batch of requests additionally including the new request for execution on the execution engine. The Accused Functionality receives a new set of requests including the new request:

```
289 // Try to get a new batch
290 if let Some((mut new_entries, new_batch, span)) =
291     queue.next_batch(min_size, token_budget).await
---
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L289-L291>).

The Accused Functionality merges the new set of requests into the previous, first batch of requests:

```
318 // Extend current batch with the new batch
319 if let Some(new_cached_batch) = new_cached_batch {
320     entries.extend(new_entries);
321     batches.push(new_cached_batch);
322 }
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L318-L322>).

The Accused Functionality then schedules a second batch of requests that also includes the new request for execution:

```
339         cached_batch = decode(&mut client, batches, &mut entries, &generation_health)
340         .instrument(next_batch_span)
341         .await;
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L339-L341>).

The Accused Functionality schedules the second batch of requests responsive to determining that the execution engine has memory available to execute the second batch of requests:

```
287         let token_budget = max_batch_total_tokens - batch_max_tokens;
288
289         // Try to get a new batch
290         if let Some((mut new_entries, new_batch, span)) =
291             queue.next_batch(min_size, token_budget).await
```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L287-L291>).

```

187         if (prefill_tokens + decode_tokens) > token_budget {
188             // Entry is over budget
189             // Add it back to the front
190             self.entries.push_front((id, entry));
191             break;
192         }

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/queue.rs#L187-L192>).

```

156     /// Overall this number should be the largest possible amount that fits the
157     /// remaining memory (after the model is loaded). Since the actual memory overhead
158     /// depends on other parameters like if you're using quantization, flash attention
159     /// or the model implementation, text-generation-inference cannot infer this number
160     /// automatically.
161     #[clap(default_value = "32000", long, env)]
162     max_batch_total_tokens: u32,

```

(<https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/launcher/src/main.rs#L156-L162>).

The Accused Functionality performs no check to validate that the inputs for the second batch of requests are the same length, and implies that requests in a batch can have an arbitrary length. (See <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/queue.rs>; <https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs>).

TGI also confirms this behavior by explaining that it performs continuous batching, which allows for new requests, of different length than the first request, to be processed in the same batch:

Text Generation Inference

Text Generation Inference is a production ready inference container developed by Hugging Face to enable easy deployment of large language models.

Its main features are:

- Continuous batching
- Token streaming using Server-Sent Events (SSE)
- Tensor Parallelism for faster inference on multiple GPUs
- Optimized transformers code using custom CUDA kernels
- Production ready logging, monitoring and tracing with Prometheus and Open Telemetry

(<https://huggingface.co/blog/falcon>; see also <https://github.com/huggingface/text-generation-inference/tree/main/router>; <https://www.anyscale.com/blog/continuous-batching-llm-inference>.)

10[F] generating, by the execution engine, a second set of output tokens by applying the transformer model to the second set of inputs for the second batch.	The Accused Functionality performs the step of generating, by the execution engine, a second set of output tokens by applying the transformer model to the second set of inputs for the second batch. The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 10[C]. (See https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/src/infer.rs#L361-L362; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/sharded_client.rs#L98-L103; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/router/client/src/client.rs#L107-L108; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L61-L62; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L634-L646; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L602; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L648-L657; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/server.py#L57-L59; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L93-L99; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L147; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L219; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/flash_causal_lm.py#L239; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L347; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L277-L278; https://github.com/huggingface/text-generation-inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L277-L278;
---	---

	inference/blob/bd3a9d8e856cb7e2122f1a09d2fb0f44b7649dad/server/text_generation_server/models/custom_modeling/flash_llama_modeling.py#L140).
Claim 11. The non-transitory computer-readable storage medium of claim 10, the operations further comprising:	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 2.
11[A] responsive to determining that a request in the first batch of requests has been completed, providing output tokens generated for the completed request to a client device as a response to the request, and	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 2.
11[B] wherein the second batch of requests includes at least one of the remaining requests from the one or more requests and the new request.	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 2[A].

Claim 12. The non-transitory computer-readable storage medium of claim 11, wherein the request is associated with a cache memory in the execution engine dedicated for storing an internal state for the request, and responsive to determining that the request has been completed, freeing the dedicated cache memory for the request in the execution engine.	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 3.
Claim 13. The non-transitory computer-readable storage medium of claim 10, wherein the input for the at least one request is at least one output token from the first set of output tokens for the at least one request, and wherein a length of the sequence of input tokens for the new request is different from a length of the at	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 4.

least one output token for the at least one request.	
Claim 14. The non-transitory computer-readable storage medium of claim 13,	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 5.
14[A] wherein the execution engine includes a cache memory for maintaining a key cache tensor for storing keys and a value cache tensor for storing values for the at least one request, and	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 5.
14[B] wherein after scheduling the second batch of requests, allocating, by the execution engine, a new cache memory dedicated to maintaining a key cache tensor and a value cache tensor for the new request.	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 5[A].

Claim 15. The non-transitory computer-readable storage medium of claim 14, wherein after generating the second set of output tokens, a length of the key cache tensor for the at least one request is different from a length of a key cache tensor for the new request, and a length of the value cache tensor for the at least one request is different from a length of a value cache tensor for the new request.	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 6.
Claim 16. The non-transitory computer-readable storage medium of claim 10, after receiving the new request from the client device, determining, by the scheduler, that there is insufficient memory available to execute the second batch of requests on a second execution engine	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 7.

different from the execution engine, and responsive to the determination for the second execution engine, determining whether the execution engine has the memory available to execute the second batch of requests.	
Claim 17. The non-transitory computer-readable storage medium of claim 10, wherein the execution engine is configured as a graphics processing unit (GPU) or a tensor processing unit (TPU).	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 8.
Claim 18. The non-transitory computer-readable storage medium of claim 10, wherein each token in the sequence of input tokens represents a text unit.	The Accused Functionality performs this step in the same fashion as described above, with respect to Claim 9.