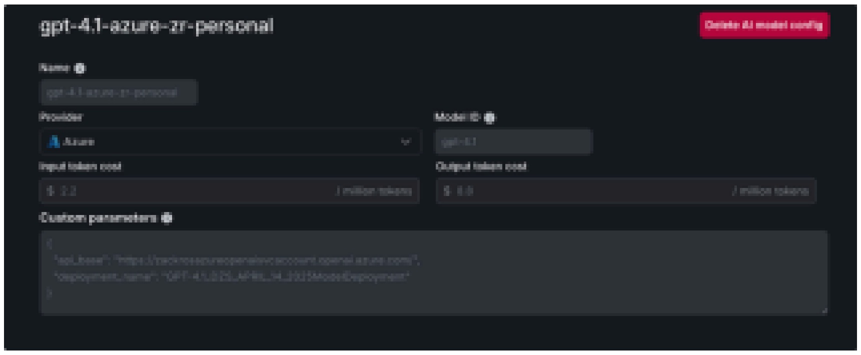
11/5/25: Evaluating Launch Darkly AI Configs

AI Model Config

Config for a particular model deployment

- endpoint
- deployment name
    - temperature (maybe – we have the option to override the temperature depending on the use case)



gpt-4.1-azure-zr-personal

Name: gpt-4.1-azure-zr-personal

Provider: Azure

Model ID: gpt-4.1

Input token cost: $ 2.2 / million tokens

Output token cost: $ 8.8 / million tokens

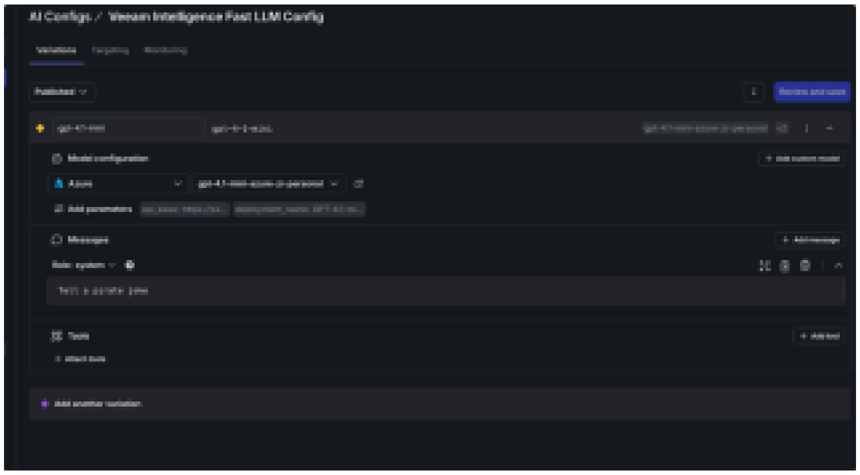
Custom parameters

```
{
  "api_base": "https://[illegible].openai.azure.com/",
  "deployment_name": "GPT-4.1_2025_APRIL_14_2025ModelDeployment"
}
```

AI Config

Config for a particular LLM-based query

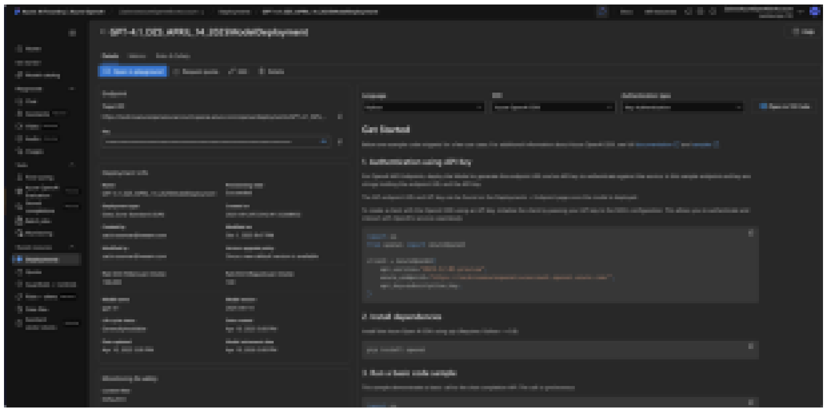
- Which AI model config to use
- Prompts to be fed to the AI model config

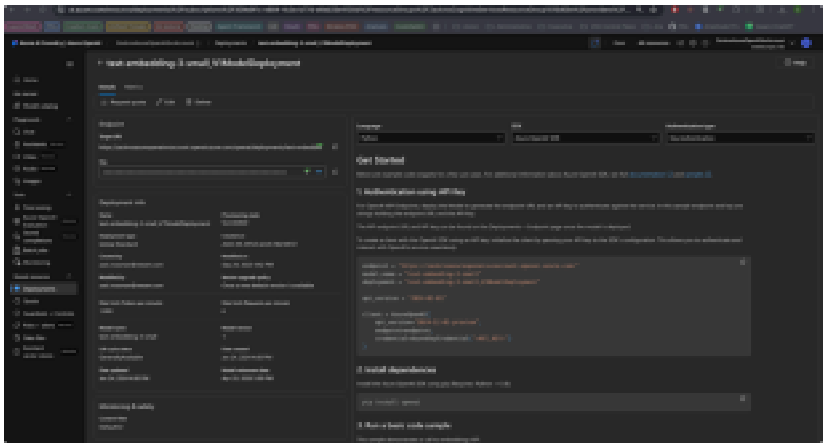


Creating AI Model Configs

First, let's create AI model configs that correspond to the model deployments in our Azure Open AI Service account.

Example model deployment details from Azure Open AI Service

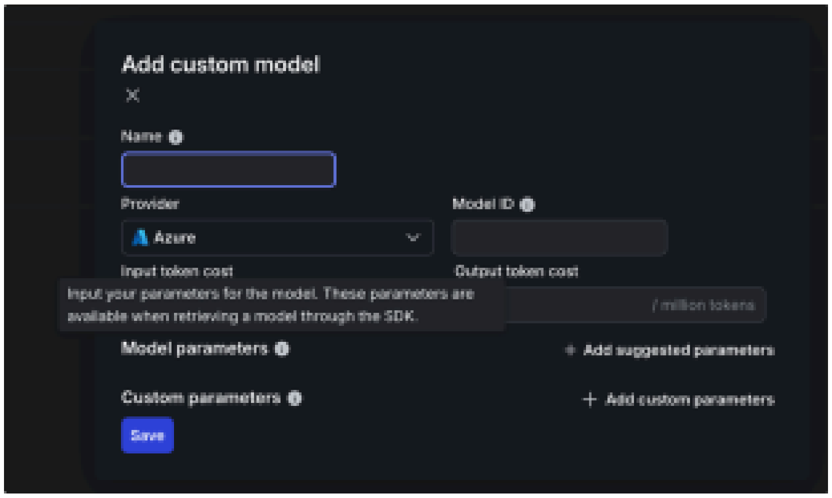




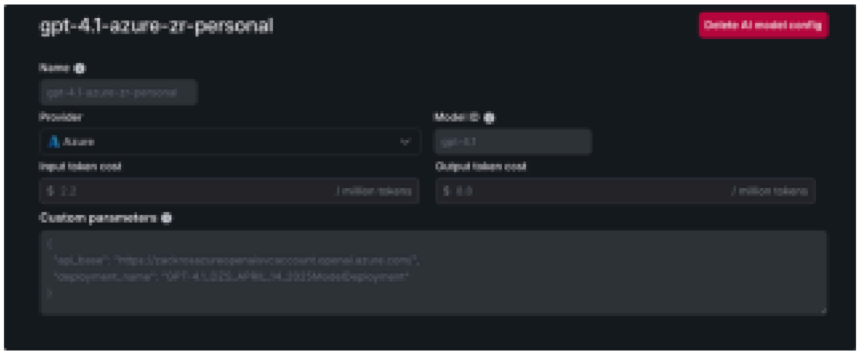
In this it seems AWS Bedrock has a standard endpoint per region and the region needs to be pulled in from the example code env vars (it's not part of the Launch Darkly Provider class. Azure Open AI Service does things different than AWS here Bedrock – each Azure Open AI Service account has its own endpoint.

We have model deployments in each VDC environment (personal, dev, stage, prod).See Azure OpenAI Configuration | Quota Allocations for Each Environment for more details. We'll need to create AI model configs for the model deployments in each env because Azure Open AI Service creates a unique endpoint for each Azure Open AI Service account. We'll add the api base endpoint, the deployment name, and maybe even the api version as a Custom Parameter.

> The endpoint and deployment_name params would probably be a better fit in Model Params rather than Custom Params because these are core params for all of our Azure LLM deployments.
>
> After talking to Launch Darkly solutions engineers, they said this was because AWS Bedrock has a consistent endpoint for all models deployed in the same region (see this).
>
> [example code]()
>
> Not ideal but at the same time, it's hard to generalize across all model providers and the Custom Params option gives us the flexibility to add the info we need for each AI Model Config.

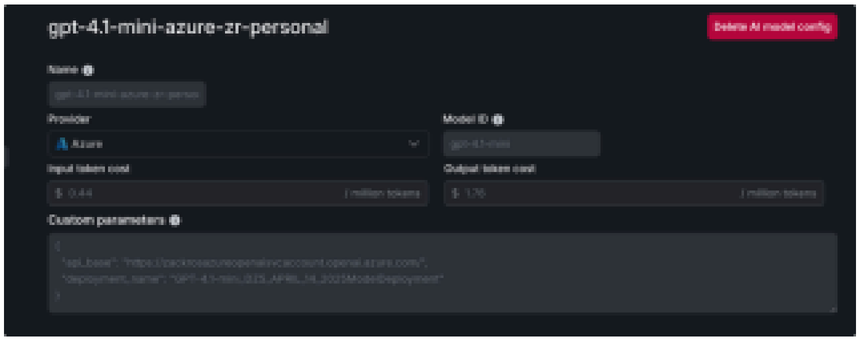


Input your parameters for the model. These parameters are available when retrieving a model through the SDK.



Name: gpt-4.1-azure-zr-personal
Provider: Azure
Model ID: gpt-4.1
Input token cost: $ 2.2 / million tokens
Output token cost: $ 8.8 / million tokens

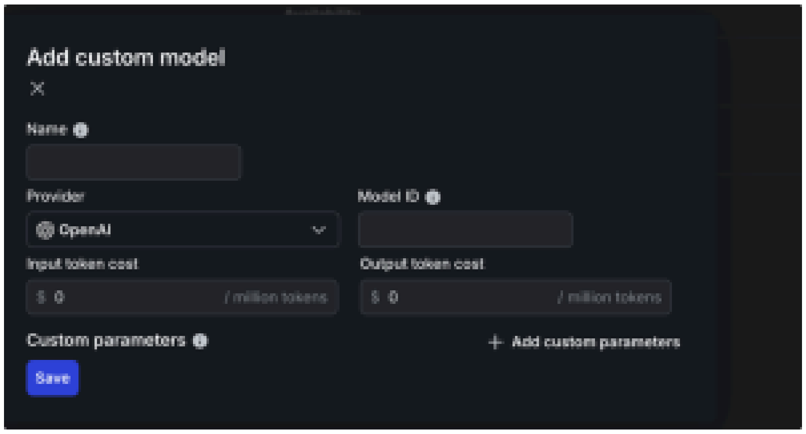
```
{
 "api_base": "https://zackroseazureopenaiinvcaccount.openai.azure.com/",
 "deployment_name": "GPT-4.1_0725_APRIL_14_2025ModelDeployment"
}
```



Name: gpt-4.1-mini-azure-zr-persor
Provider: Azure
Model ID: gpt-4.1-mini
Input token cost: $ 0.44 / million tokens
Output token cost: $ 1.76 / million tokens

```
{
 "api_base": "https://zackroseazureopenaiinvcaccount.openai.azure.com/",
 "deployment_name": "GPT-4.1-mini_0725_APRIL_14_2025ModelDeployment"
}
```

> Some AI model providers support "model params" and "custom params" (e.g. Azure) while others only support "custom params" (e.g. Open AI)?


Add custom model

Name

Provider: OpenAI

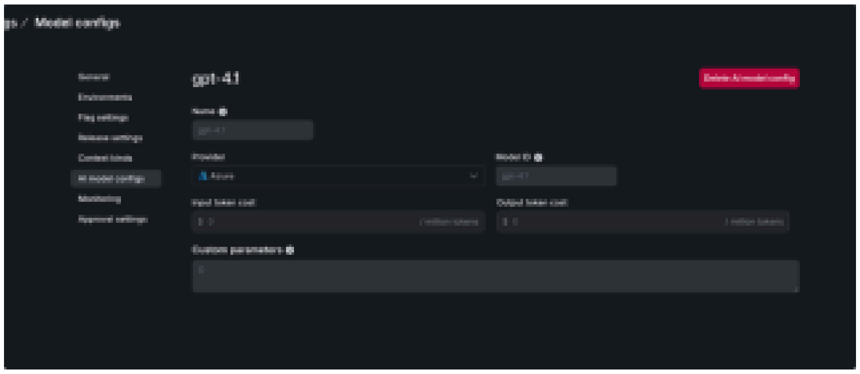
Model ID

Input token cost: $ 0 / million tokens

Output token cost: $ 0 / million tokens

Custom parameters

(minor feedback) why can’t we edit the Model Config after creation? Have to delete and recreate if we want to change the name, token input/output cost, or the custom params


gs / Model configs

gpt-4.1

Name: gpt-4.1

Provider: Azure

Model ID: gpt-4.1

Input token cost: $ 0 / million tokens

Output token cost: $ 0 / million tokens

Custom parameters: {}

Creating AI Configs

I created two “completion based” AI config (rather than “AI agent” AI config) because this is the best fit for the Veeam Intelligence workflow.

Details on "Completion-Based" AI configs vs. "Agent-Based" AI configs

Quickstart for AI Configs | LaunchDarkly | Documentation

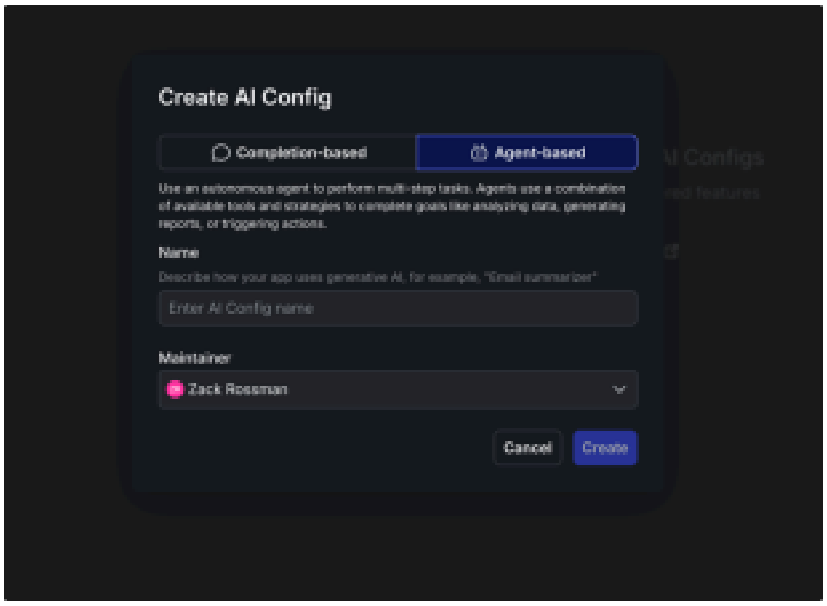
AI Configs can be created in two modes:

Prompt-based: Configure prompts with messages and roles.

Agent-based: Configure multi-step workflows with instructions and tools. To learn more, read Agents in AI Configs.

Agents in AI Configs | LaunchDarkly | Documentation

Use agent-based AI Configs to define, monitor, and adapt structured model workflows in real time. Agents use the same configuration model as standard AI Configs, including environment targeting, variation management, and monitoring. The difference is that agents define how a model performs reasoning and takes actions across multiple steps instead of responding to a single prompt.


Create AI Config

Completion-based | Agent-based

Use an autonomous agent to perform multi-step tasks. Agents use a combination of available tools and strategies to complete goals like analyzing data, generating reports, or triggering actions.

Name

Describe how your app uses generative AI, for example, "Email summarizer"

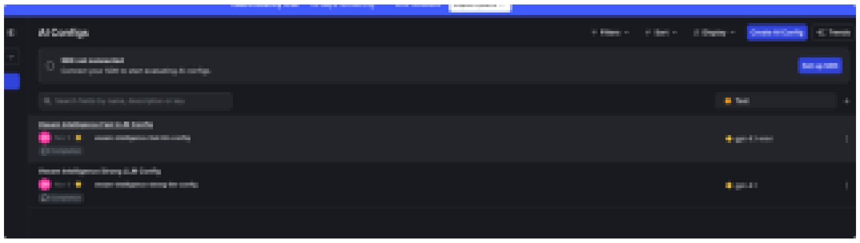
Enter AI Config name

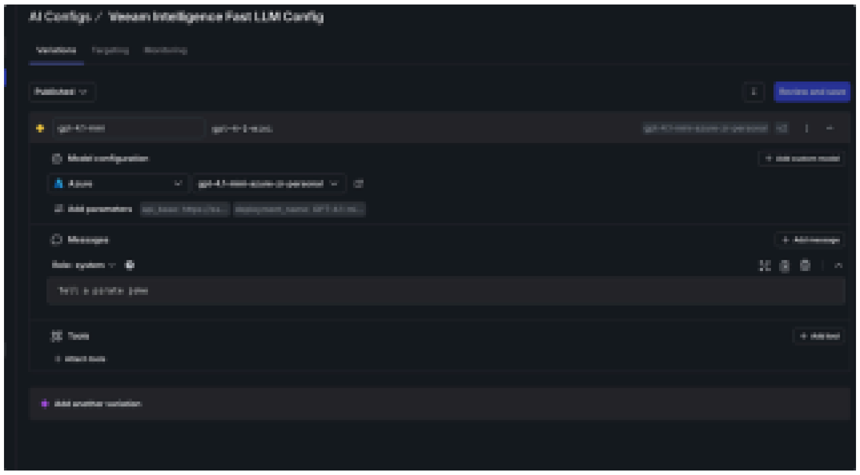
Maintainer

Zack Rossman

Cancel | Create

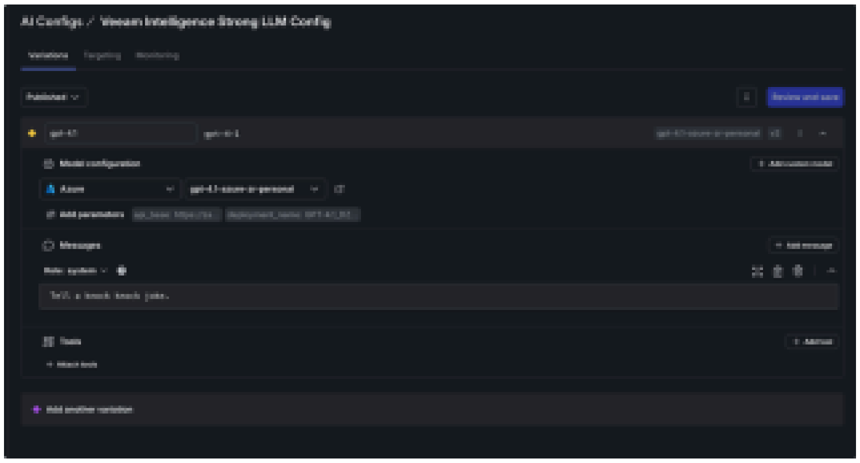
One AI config for the “strong” model for handling more complex queries, and another for the “fast” model for handling simpler queries.




AI Configs / Veeam Intelligence Fast LLM Config

gpt-4.1-mini-azure-ai-personal

Tell a pirate joke

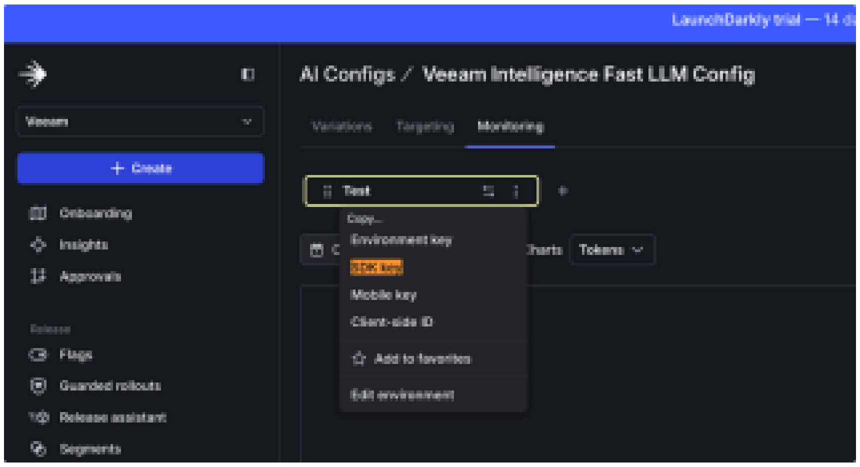

AI Configs / Veeam Intelligence Strong LLM Config

gpt-4.1-azure-ai-personal

Tell a knock knock joke.

Get SDK Key

This was tricky. At first I thought the application code needed to use an API key to fetch the AI configs. But actually we need to use an SDK key to fetch the AI configs.


AI Configs / Veeam Intelligence Fast LLM Config

Tracking Metrics

I added a wrapper to Veeam Intelligence to track various metrics

Generation count - Number of successful completions

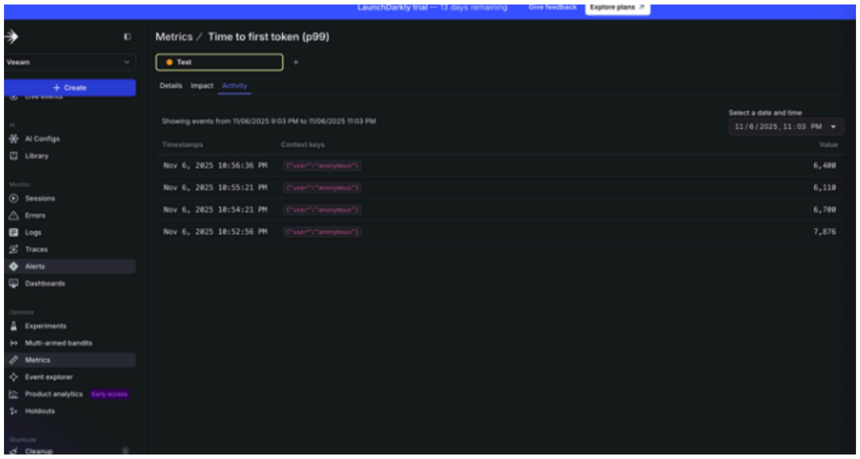
Input tokens - Tokens sent to model

Output tokens - Tokens generated by model

Duration - Time to generate

Time to first token - Latency metric

Errors - Failed generations



LaunchDarkly trial — 13 days remaining   Give feedback   Explore plans

Metrics / Time to first token (p99)

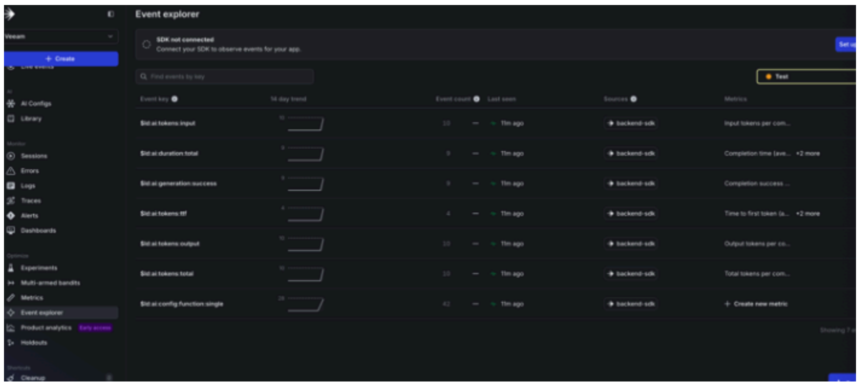
Test

Details   Impact   Activity

Showing events from 11/06/2025 9:03 PM to 11/06/2025 11:03 PM

Select a date and time: 11/6/2025, 11:03 PM

Timestamps	Context keys	Value
Nov 6, 2025 10:56:36 PM	{"user":"anonymous"}	6,400
Nov 6, 2025 10:55:21 PM	{"user":"anonymous"}	6,110
Nov 6, 2025 10:54:21 PM	{"user":"anonymous"}	6,700
Nov 6, 2025 10:52:56 PM	{"user":"anonymous"}	7,876



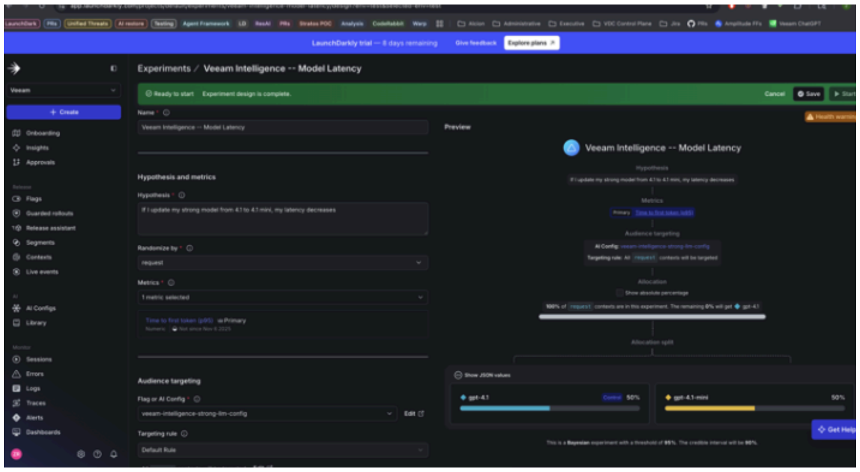
Event explorer

SDK not connected
Connect your SDK to observe events for your app.

Event key	14 day trend	Event count	Last seen	Sources	Metrics
$ld:ai:tokens:input	10	10	11m ago	backend-sdk	Input tokens per com...
$ld:ai:duration:total	9	9	11m ago	backend-sdk	Completion time (ave... +2 more
$ld:ai:generation:success	9	9	11m ago	backend-sdk	Completion success ...
$ld:ai:tokens:ttf	4	4	11m ago	backend-sdk	Time to first token (a... +2 more
$ld:ai:tokens:output	10	10	11m ago	backend-sdk	Output tokens per co...
$ld:ai:tokens:total	10	10	11m ago	backend-sdk	Total tokens per com...
$ld:ai:config:function:single	28	42	11m ago	backend-sdk	+ Create new metric

Experiments

I created an experiment to understand – can we quantify the difference in “time to first token” metric if we use a lighter weight model like GPT 4.1-mini rather than GPT 4.1?


Experiments / Veeam Intelligence -- Model Latency

Ready to start Experiment design is complete.

Name: Veeam Intelligence -- Model Latency

Hypothesis and metrics

Hypothesis: If I update my strong model from 4.1 to 4.1 mini, my latency decreases

Randomize by: request

Metrics: 1 metric selected

Time to first token (p95) Primary

Audience targeting

Flag or AI Config: veeam-intelligence-strong-llm-config

Targeting rule: Default Rule

Preview

Veeam Intelligence -- Model Latency

Hypothesis: If I update my strong model from 4.1 to 4.1 mini, my latency decreases

Metrics: Primary Time to first token (p95)

Audience targeting: AI Config: veeam-intelligence-strong-llm-config. Targeting rule: All request contexts will be targeted

Allocation: 100% of request contexts are in this experiment. The remaining 0% will get gpt-4.1

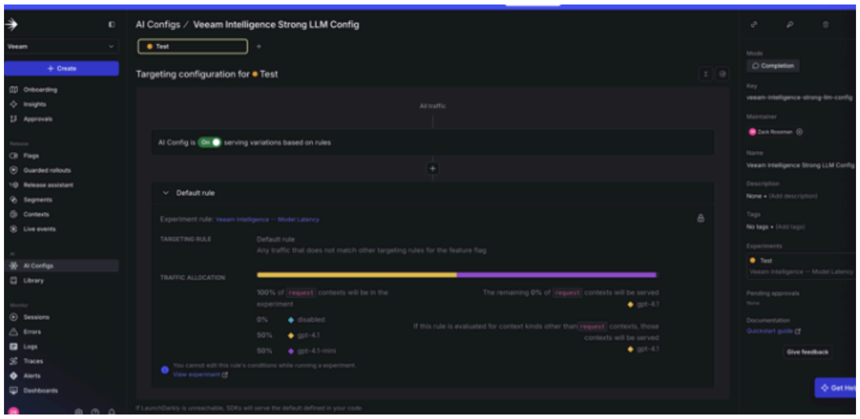
Allocation split

gpt-4.1 Control 50%

gpt-4.1-mini 50%

This is a Bayesian experiment with a threshold of 95%. The credible interval will be 90%.

Once the experiment was started, the targeting rules for my “strong” AI config were automatically updated according to the 50/50 allocation split for my experiment.


AI Configs / Veeam Intelligence Strong LLM Config

Targeting configuration for Test

All traffic

AI Config is On serving variations based on rules

Default rule

Experiment rule: Veeam Intelligence -- Model Latency

TARGETING RULE: Default rule. Any traffic that does not match other targeting rules for the feature flag

TRAFFIC ALLOCATION: 100% of request contexts will be in the experiment

0%	disabled
50%	gpt-4.1
50%	gpt-4.1-mini

The remaining 0% of request contexts will be served gpt-4.1

If this rule is evaluated for context kinds other than request contexts, those contexts will be served gpt-4.1

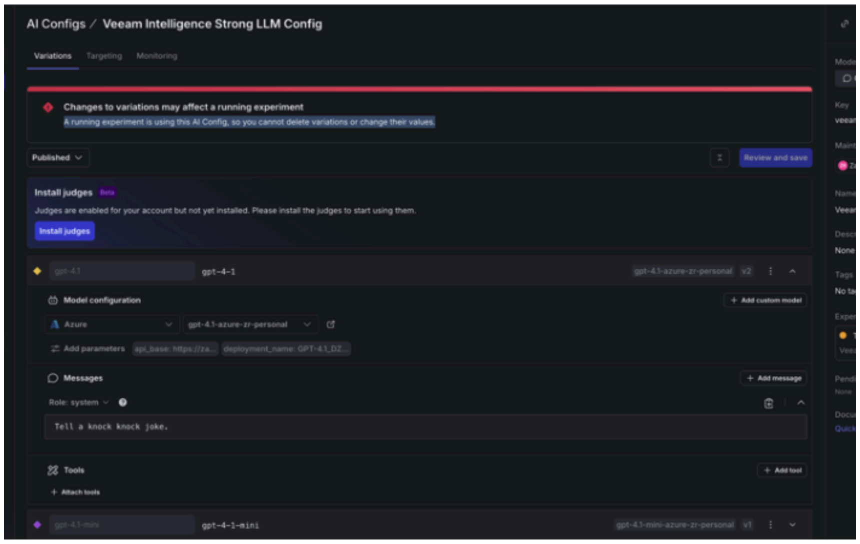
You cannot edit this rule's conditions while running a experiment. View experiment

If LaunchDarkly is unreachable, SDKs will serve the default defined in your code.

Key: veeam-intelligence-strong-llm-config

Name: Veeam Intelligence Strong LLM Config

Note – when an AI config is being used as part of an experiment, you can’t delete variations or change their value. This is smart, don’t want to subvert the results


AI Configs / Veeam Intelligence Strong LLM Config

Changes to variations may affect a running experiment

A running experiment is using this AI Config, so you cannot delete variations or change their values.

Install judges

Judges are enabled for your account but not yet installed. Please install the judges to start using them.

gpt-4.1 gpt-4-1 gpt-4.1-azure-zr-personal v2

Model configuration: Azure gpt-4.1-azure-zr-personal

Add parameters: api_base: https://za... deployment_name: GPT-4.1_DZ...

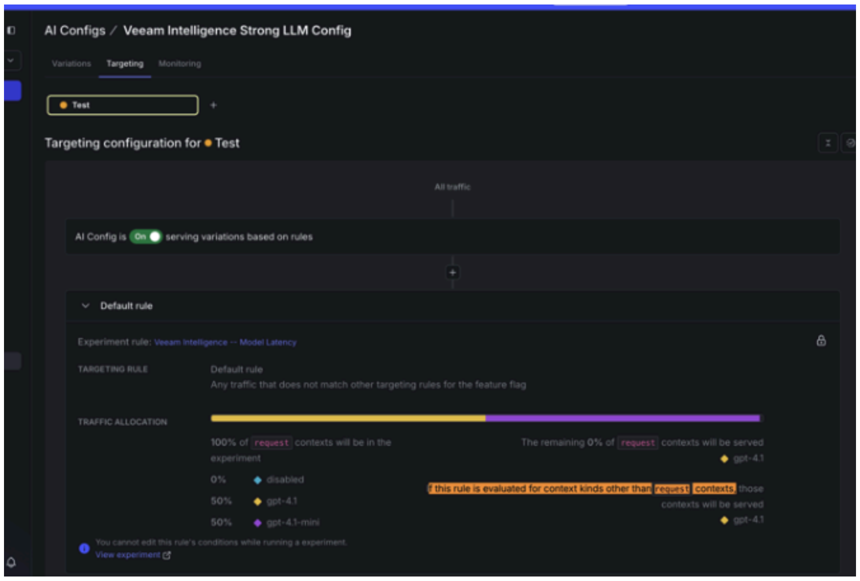
Messages

Role: system

```
Tell a knock knock joke.
```

gpt-4.1-mini gpt-4-1-mini gpt-4.1-mini-azure-zr-personal v1

At first, I noticed that the experiment wasn’t tracking the results for requests and the AI config variations weren’t being distributed 50/50. Rather, all 100% of requests were getting the gpt-4.1 model config. The issue was nuanced – my code wasn’t using the correct “context kind”


AI Configs / Veeam Intelligence Strong LLM Config

Targeting configuration for ● Test

AI Config is On serving variations based on rules

Default rule

Experiment rule: Veeam Intelligence -- Model Latency

TARGETING RULE — Default rule
Any traffic that does not match other targeting rules for the feature flag

TRAFFIC ALLOCATION

100% of request contexts will be in the experiment

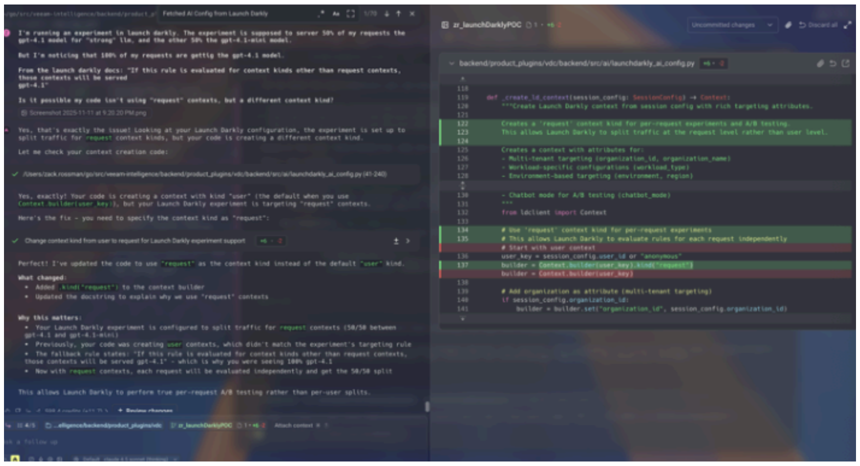
0% ◆ disabled
50% ◆ gpt-4.1
50% ◆ gpt-4.1-mini

The remaining 0% of request contexts will be served ◆ gpt-4.1

If this rule is evaluated for context kinds other than request contexts, those contexts will be served ◆ gpt-4.1

You cannot edit this rule's conditions while running a experiment.
View experiment

I updated my code to use the request context kind


Fetched AI Config from Launch Darkly

I'm running an experiment in launch darkly. The experiment is supposed to server 50% of my requests the gpt-4.1 model for "strong" llm, and the other 50% the gpt-4.1-mini model.

But I'm noticing that 100% of my requests are gettig the gpt-4.1 model.

From the launch darkly docs: "If this rule is evaluated for context kinds other than request contexts, those contexts will be served
gpt-4.1"

Is it possible my code isn't using "request" contexts, but a different context kind?

Screenshot 2025-11-11 at 9.20.20 PM.png

Yes, that's exactly the issue! Looking at your Launch Darkly configuration, the experiment is set up to split traffic for request context kinds, but your code is creating a different context kind.

Let me check your context creation code:

✓ /Users/zack.rossman/go/src/veeam-intelligence/backend/product_plugins/vdc/backend/src/ai/launchdarkly_ai_config.py (41-240)

Yes, exactly! Your code is creating a context with kind "user" (the default when you use Context.builder(user_key)), but your Launch Darkly experiment is targeting "request" contexts.

Here's the fix - you need to specify the context kind as "request":

✓ Change context kind from user to request for Launch Darkly experiment support +6 -2

Perfect! I've updated the code to use "request" as the context kind instead of the default "user" kind.

What changed:
- Added .kind("request") to the context builder
- Updated the docstring to explain why we use "request" contexts

Why this matters:
- Your Launch Darkly experiment is configured to split traffic for request contexts (50/50 between gpt-4.1 and gpt-4.1-mini)
- Previously, your code was creating user contexts, which didn't match the experiment's targeting rule
- The fallback rule states: "If this rule is evaluated for context kinds other than request contexts, those contexts will be served gpt-4.1" - which is why you were seeing 100% gpt-4.1
- Now with request contexts, each request will be evaluated independently and get the 50/50 split

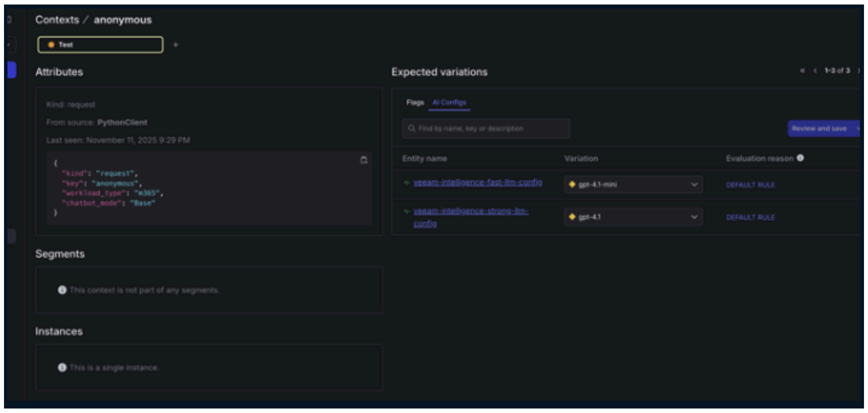
This allows Launch Darkly to perform true per-request A/B testing rather than per-user splits.

zr_launchDarklyPOC

backend/product_plugins/vdc/backend/src/ai/launchdarkly_ai_config.py +6 -2

```

def _create_ld_context(session_config: SessionConfig) -> Context:
    """Create Launch Darkly context from session config with rich targeting attributes.

    Creates a 'request' context kind for per-request experiments and A/B testing.
    This allows Launch Darkly to split traffic at the request level rather than user level.

    Creates a context with attributes for:
    - Multi-tenant targeting (organization_id, organization_name)
    - Workload-specific configurations (workload_type)
    - Environment-based targeting (environment, region)

    - Chatbot mode for A/B testing (chatbot_mode)
    """
    from ldclient import Context

    # Use 'request' context kind for per-request experiments
    # This allows Launch Darkly to evaluate rules for each request independently
    # Start with user context
    user_key = session_config.user_id or "anonymous"
    builder = Context.builder(user_key).kind("request")
    builder = Context.builder(user_key)

    # Add organization as attribute (multi-tenant targeting)
    if session_config.organization_id:
        builder = builder.set("organization_id", session_config.organization_id)
```

Hmmm… but even with the context kind set to request , Launch Darkly was still serving the default gpt-4.1


Contexts / anonymous

● Test

Attributes

Kind: request
From source: PythonClient
Last seen: November 11, 2025 9:29 PM

```
{
  "kind": "request",
  "key": "anonymous",
  "workload_type": "m365",
  "chatbot_mode": "Base"
}
```

Segments

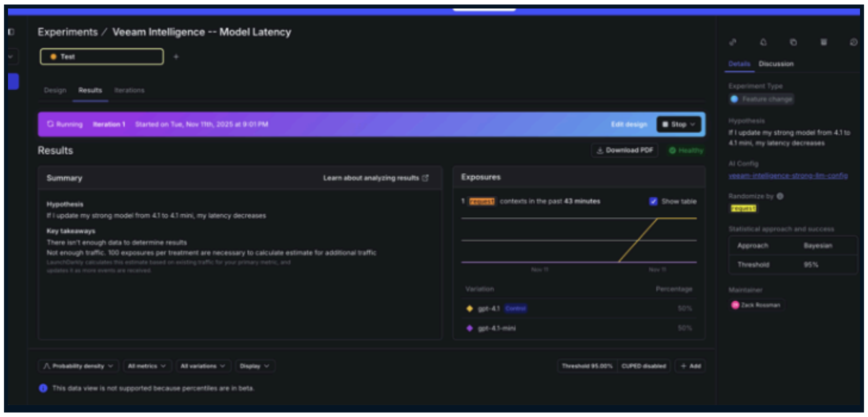
This context is not part of any segments.

Instances

This is a single instance.

Expected variations

Entity name	Variation	Evaluation reason
veeam-intelligence-fast-llm-config	◆ gpt-4.1-mini	DEFAULT RULE
veeam-intelligence-strong-llm-config	◆ gpt-4.1	DEFAULT RULE



Results

Summary — Learn about analyzing results

Hypothesis
If I update my strong model from 4.1 to 4.1 mini, my latency decreases

Key takeaways
There isn't enough data to determine results
Not enough traffic. 100 exposures per treatment are necessary to calculate estimate for additional traffic
LaunchDarkly calculates this estimate based on existing traffic for your primary metric, and updates it as more events are received.

Exposures

1 request contexts in the past 43 minutes — Show table

Variation	Percentage
gpt-4.1 Control	50%
gpt-4.1-mini	50%

Experiment Type: Feature change

Hypothesis: If I update my strong model from 4.1 to 4.1 mini, my latency decreases

AI Config: veeam-intelligence-strong-llm-config

Randomize by: request

Statistical approach and success

Approach	Bayesian
Threshold	95%

Maintainer: Zack Rossman

This data view is not supported because percentiles are in beta.

Debugging the issue with Warp AI agent

I see the issue! The context key is always "anonymous" (or the same user_id), which means Launch Darkly sees it as the same context across all requests. For per-request randomization in experiments, you need a unique key per request.

The experiment's traffic allocation uses the context key to determine which variation to serve. If the key is always the same, it will always hash to the same variation (gpt-4.1 in your case).

Here's the fix - generate a unique key per request:
Perfect! Now each request will have a unique UUID as the context key, which allows Launch Darkly's traffic allocation algorithm to properly randomize between the experiment variations.

What changed:

1. Unique key per request: Now using uuid.uuid4() to generate a unique key for each request
2. user_id as attribute: Moved user_id from the key to an attribute, so you can still target by user if needed in the future
3. Proper randomization: With unique keys, Launch Darkly can properly hash and distribute traffic according to your 50/50 experiment split

Why this fixes it:

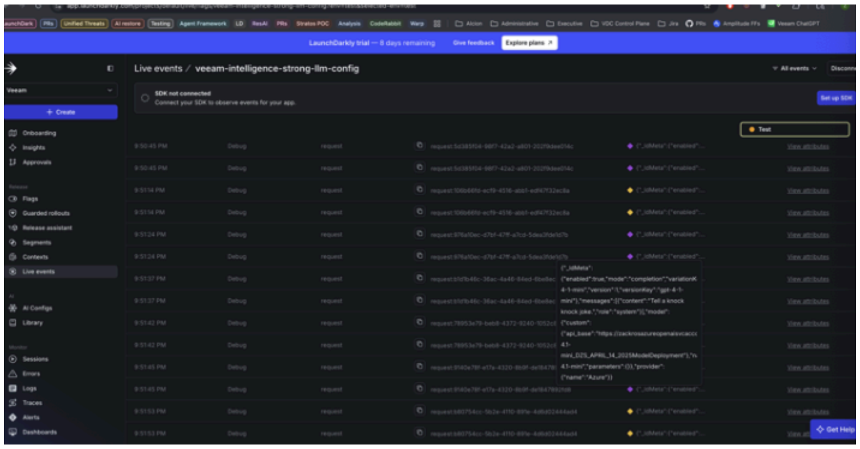
Launch Darkly's traffic allocation uses the context key to hash and determine which variation to serve

When you were using the same key ("anonymous") for all requests, it always hashed to the same bucket → always gpt-4.1

With unique keys per request, each request gets independently randomized → true 50/50 split

Now your experiment should properly distribute traffic between gpt-4.1 and gpt-4.1-mini!

The issue was I needed to add a unique requestID for each AI config evaluation. This fixes it and, indeed, now we can see an even split between requests

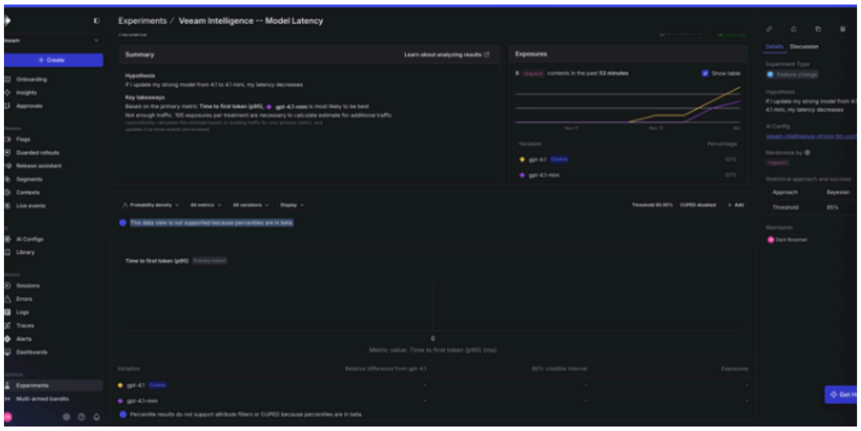


Live events / veeam-intelligence-strong-llm-config

SDK not connected
Connect your SDK to observe events for your app.

```
{"_ldMeta":
{"enabled":true,"mode":"completion","variationKey":"gpt-4-1-mini","version":1,"versionKey":"gpt-4-1-mini"},"messages":[{"content":"Tell a knock knock joke.","role":"system"}],"model":
{"custom":
{"api_base":"https://zackrosazureopenaisvcaccc
4.1-
mini_D25_APRIL_14_2025ModelDeployment"},"name":"gpt-
4.1-mini","parameters":{}},"provider":
{"name":"Azure"}}
```

Unfortunately, it seems the result view and the CUPED adjustment isn't supported for percentile analysis like p95



Experiments / Veeam Intelligence -- Model Latency

Summary

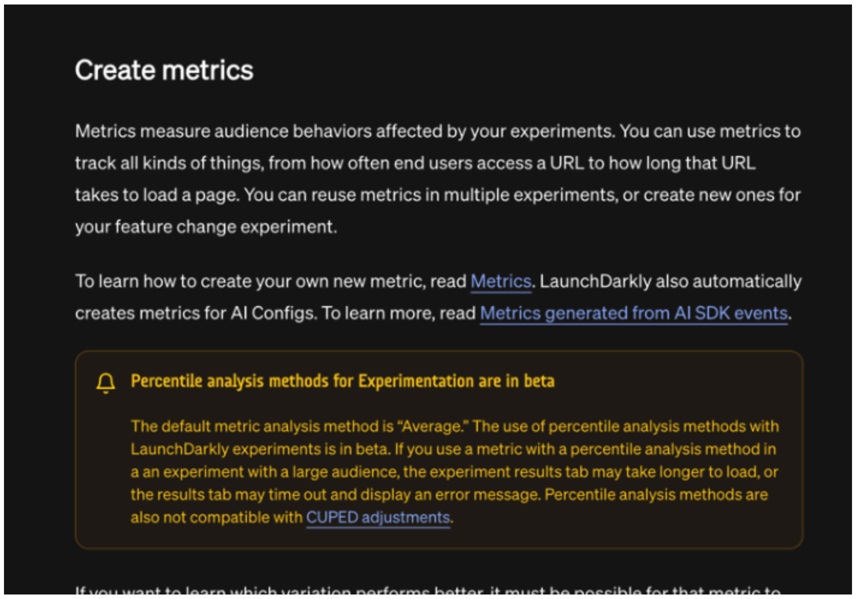
Hypothesis
If I update my strong model from 4.1 to 4.1 mini, my latency decreases

Key takeaways
Based on the primary metric Time to first token (p95), gpt-4.1-mini is most likely to be best
Not enough traffic. 100 exposures per treatment are necessary to calculate estimate for additional traffic

This data view is not supported because percentiles are in beta.

Time to first token (p95)

Percentile results do not support attribute filters or CUPED because percentiles are in beta.



Create metrics

Metrics measure audience behaviors affected by your experiments. You can use metrics to track all kinds of things, from how often end users access a URL to how long that URL takes to load a page. You can reuse metrics in multiple experiments, or create new ones for your feature change experiment.

To learn how to create your own new metric, read Metrics. LaunchDarkly also automatically creates metrics for AI Configs. To learn more, read Metrics generated from AI SDK events.

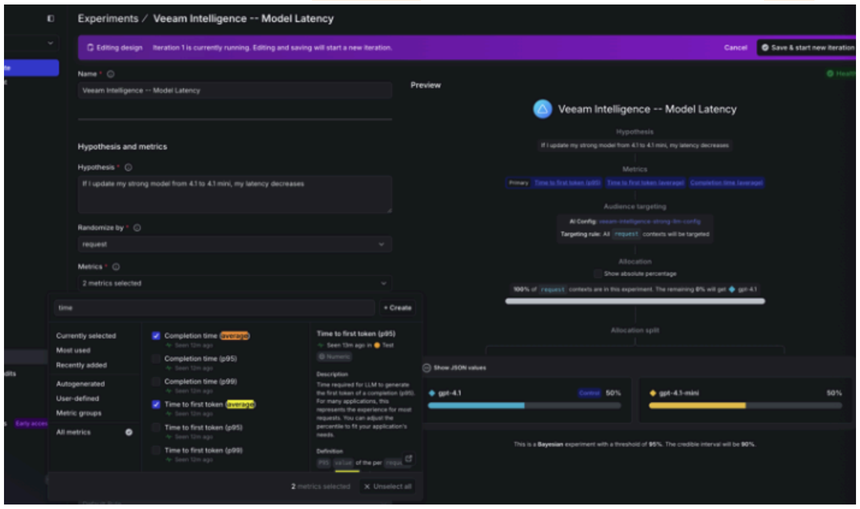
Percentile analysis methods for Experimentation are in beta

The default metric analysis method is "Average." The use of percentile analysis methods with LaunchDarkly experiments is in beta. If you use a metric with a percentile analysis method in a an experiment with a large audience, the experiment results tab may take longer to load, or the results tab may time out and display an error message. Percentile analysis methods are also not compatible with CUPED adjustments.

If you want to learn which variation performs better, it must be possible for that metric to

Creating feature change experiments | LaunchDarkly | Documentation

So I recreated the experiment where we're measuring average time to first token (rather than p95 time to first token)



Experiments / Veeam Intelligence -- Model Latency

Editing design Iteration 1 is currently running. Editing and saving will start a new iteration.

Name: Veeam Intelligence -- Model Latency

Hypothesis and metrics

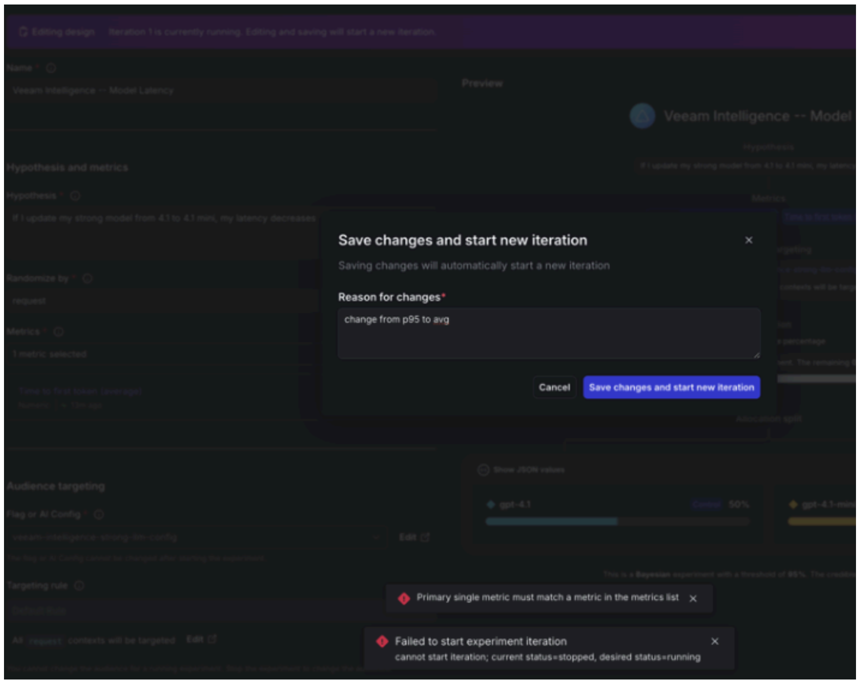
Hypothesis: If I update my strong model from 4.1 to 4.1 mini, my latency decreases

Randomize by: request

Metrics: 2 metrics selected

Cancel
Save & start new iteration
time
+ Create
Currently selected
Most used
Recently added
Autogenerated
User-defined
Metric groups
All metrics
Completion time (average)
Completion time (p95)
Completion time (p99)
Time to first token (average)
Time to first token (p95)
Time to first token (p99)
Time to first token (p95)
Description
Time required for LLM to generate the first token of a completion (p95). For many applications, this represents the experience for most requests. You can adjust the percentile to fit your application's needs.
2 metrics selected
Unselect all
Preview
Veeam Intelligence -- Model Latency
Hypothesis
Metrics
Primary
Time to first token (p95)
Time to first token (average)
Completion time (average)
Audience targeting
AI Config: veeam-intelligence-strong-llm-config
Targeting rule: All request contexts will be targeted
Allocation
Show absolute percentage
100% of request contexts are in this experiment. The remaining 0% will get gpt-4.1
Allocation split
Show JSON values
gpt-4.1 Control 50%
gpt-4.1-mini 50%
This is a Bayesian experiment with a threshold of 95%. The credible interval will be 90%.

I tried to create a new "iteration" of the existing experiment, but I had to start a new experiment altogether because the "primary single metric must match a metric in the metrics list". So the issue was that I created the initial experiment with a percentage-based metric as the primary metric.



Save changes and start new iteration

Saving changes will automatically start a new iteration

Reason for changes*

change from p95 to avg

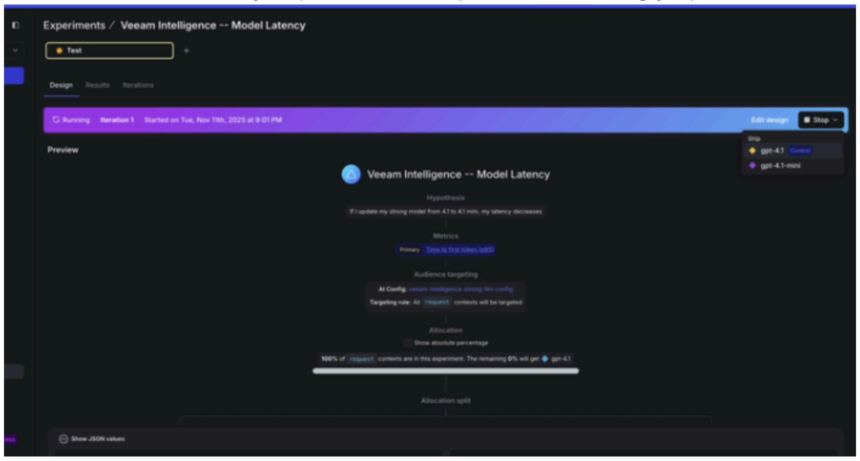
Primary single metric must match a metric in the metrics list

Failed to start experiment iteration

cannot start iteration; current status=stopped, desired status=running

When you stop an experiment, Launch Darkly asks to “ship” a single variation. Is this a “feature” or a “bug”? I like it because it forces us to be decisive.

This is one example of how the Launch Darkly experimentation platform is strongly opinionated.



Experiments / Veeam Intelligence -- Model Latency

Running Iteration 1 Started on Tue, Nov 11th, 2025 at 9:01 PM

Preview

Veeam Intelligence -- Model Latency

Hypothesis

If I update my strong model from 4.1 to 4.1 mini, my latency decreases

Metrics

Primary Time to first token (p95)

Audience targeting

AI Config: veeam-intelligence-strong-llm-config

Targeting rule: All request contexts will be targeted

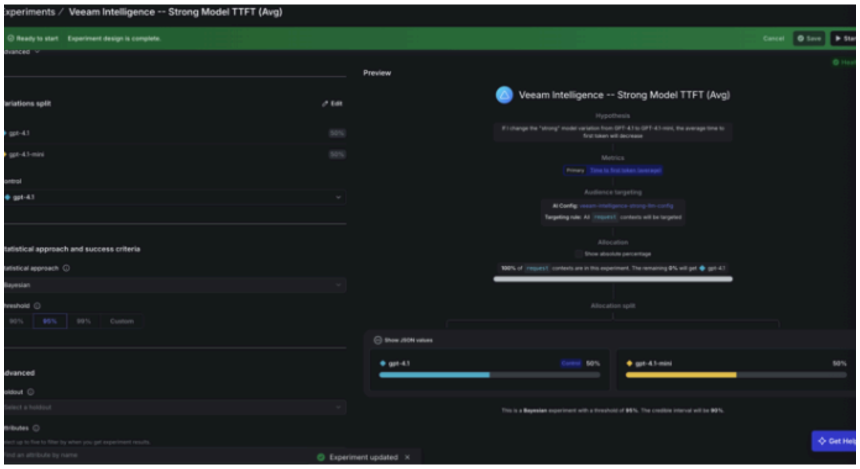
Allocation

Show absolute percentage

100% of request contexts are in this experiment. The remaining 0% will get gpt-4.1

Allocation split

I recreated the experiment with time to first token (average) as the primary metric



Experiments / Veeam Intelligence -- Strong Model TTFT (Avg)

Ready to start Experiment design is complete.

Variations split

gpt-4.1	50%
gpt-4.1-mini	50%

Control: gpt-4.1

Statistical approach and success criteria

Statistical approach: Bayesian

Threshold: 90% [✓ 95%] 99% Custom

Advanced

Holdout: Select a holdout

Attributes: Select up to five to filter by when you get experiment results.

Preview

Veeam Intelligence -- Strong Model TTFT (Avg)

Hypothesis: If I change the "strong" model variation from GPT-4.1 to GPT-4.1-mini, the average time to first token will decrease

Metrics: Primary Time to first token (average)

Audience targeting: AI Config: veeam-intelligence-strong-llm-config Targeting rule: All request contexts will be targeted

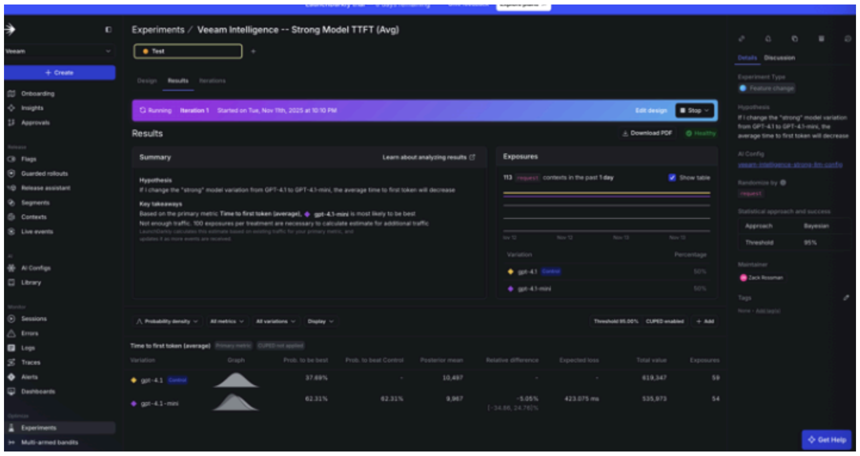
Allocation: 100% of request contexts are in this experiment. The remaining 0% will get gpt-4.1

Allocation split: gpt-4.1 Control 50% | gpt-4.1-mini 50%

This is a Bayesian experiment with a threshold of 95%. The credible interval will be 90%.

Experiment updated

They have a nice “health check” feature in on the right hand side of the experiment UI. This is a really thoughtful feature – helps us build confidence in our experiment setup and operations



Experiments / Veeam Intelligence -- Strong Model TTFT (Avg)

Running Iteration 1 Started on Tue, Nov 11th, 2025 at 10:10 PM

Results Healthy

Summary

Hypothesis
If I change the "strong" model variation from GPT-4.1 to GPT-4.1-mini, the average time to first token will decrease

Key takeaways
Based on the primary metric **Time to first token (average)**, **gpt-4.1-mini** is most likely to be best
Not enough traffic. 100 exposures per treatment are necessary to calculate estimate for additional traffic
LaunchDarkly calculates this estimate based on existing traffic for your primary metric, and updates it as more events are received.

Exposures
113 request contexts in the past 1 day

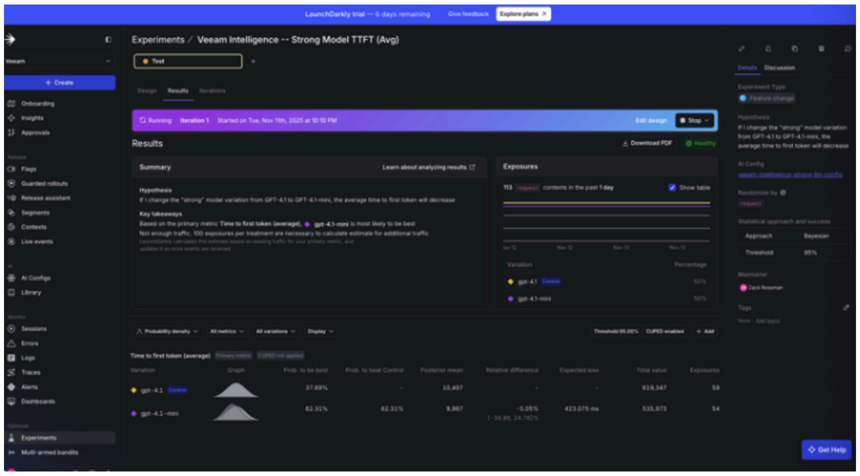
Variation	Percentage
gpt-4.1 Control	50%
gpt-4.1-mini	50%

Threshold 95.00% CUPED enabled

Time to first token (average) Primary metric CUPED not applied

Variation	Prob. to be best	Prob. to beat Control	Posterior mean	Relative difference	Expected loss	Total value	Exposures
gpt-4.1 Control	37.69%	-	10,497	-	-	619,347	59
gpt-4.1-mini	62.31%	62.31%	9,967	-5.05% [-34.86, 24.76]%	423.075 ms	535,973	54

After 115 requests, 59 requests were handled by the gpt-4.1 variation and 54 requests were handled by the gpt-4.1- mini variation. We can see there’s a 62% chance that gpt-4.1-mini is the superior variation if we want to pick the variation that reduces the average time to first token



Experiments / Veeam Intelligence -- Strong Model TTFT (Avg)

Running Iteration 1 Started on Tue, Nov 11th, 2025 at 10:10 PM

Results Healthy

Hypothesis
If I change the "strong" model variation from GPT-4.1 to GPT-4.1-mini, the average time to first token will decrease

Key takeaways
Based on the primary metric Time to first token (average), gpt-4.1-mini is most likely to be best
Not enough traffic. 100 exposures per treatment are necessary to calculate estimate for additional traffic

Variation	Prob. to be best	Prob. to beat Control	Posterior mean	Relative difference	Expected loss	Total value	Exposures
gpt-4.1 Control	37.69%	-	10,497	-	-	619,347	59
gpt-4.1-mini	62.31%	62.31%	9,967	-5.05% [-34.86, 24.76]%	423.075 ms	535,973	54

Evaluations

Online evals in AI Configs | LaunchDarkly | Documentation

Online evals use AI judges, which are a type of AI Config that evaluates another AI Config's output in real time. Judges apply a consistent evaluation prompt and scoring framework to measure each response on key metrics such as accuracy, relevance, and toxicity.

Scores appear automatically on the Monitoring tab for each variation, alongside latency, cost, and satisfaction metrics. This provides a continuous signal of model performance with real users and data, enabling teams to detect regressions, improve reliability, and apply guardrails within the AI Config workflow.

Online evals differ from offline or pre-deployment testing. Instead of running evaluations manually in a sandbox or against datasets, they measure AI Config quality continuously in production.

Online evals work alongside observability. Observability helps you view model responses and routing details, while online evals provide quality scores you can use to trigger alerts, manage guarded rollouts, or run experiments.

Use online evals to:

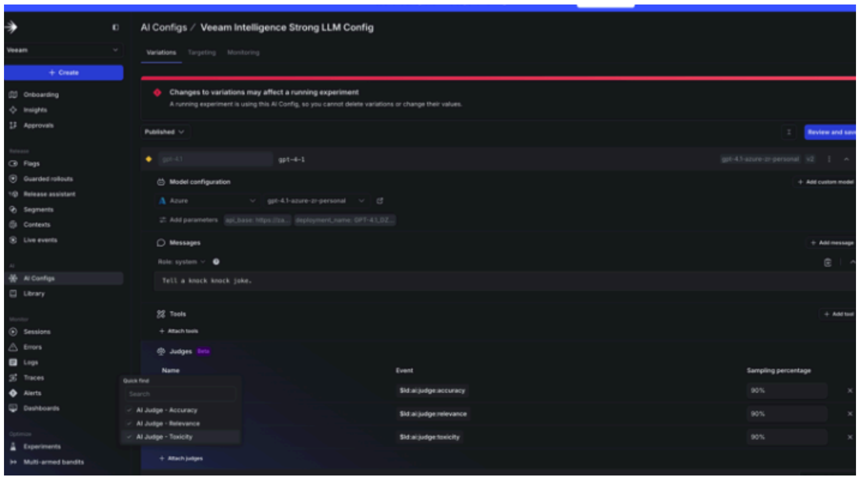
Continuously monitor AI quality in production

Detect regressions immediately after a rollout

Automate rollbacks or alerts based on evaluation metrics

Compare prompt or model variations using live performance data

I attached all three judges to my AI configs



Changes to variations may affect a running experiment
A running experiment is using this AI Config, so you cannot delete variations or change their values.

gpt-4-1

Azure   gpt-4.1-azure-zr-personal

```
Tell a knock knock joke.
```

Name	Event	Sampling percentage
AI Judge - Accuracy	$ld:ai:judge:accuracy	90%
AI Judge - Relevance	$ld:ai:judge:relevance	90%
AI Judge - Toxicity	$ld:ai:judge:toxicity	90%

The Launch Darkly judge evaluation capability is only supported in the_. This is a bummer because we primarily JS SDK use Python and Go for our backend AI services. But this is a temporary limitation, estimated that the Python SDK will have support within 3 weeks (by Dec 8, 2025).

Observability

[LLM observability | LaunchDarkly | Documentation]

☐ TODO

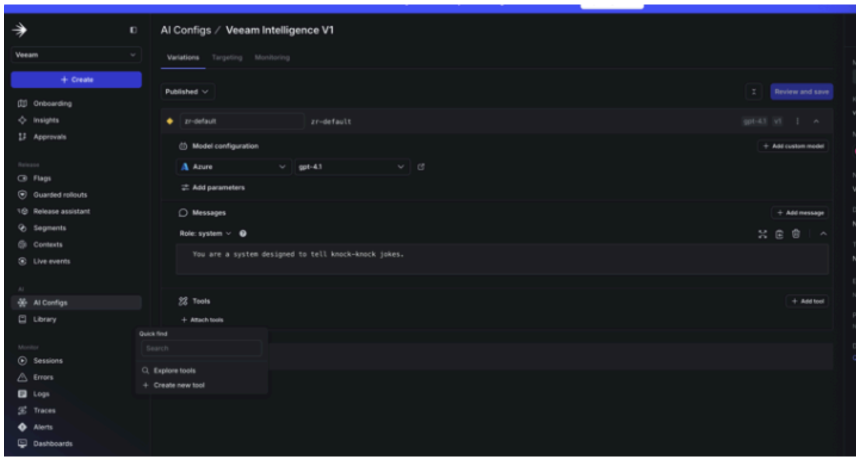
Tracing

☐ TODO

Creating Tools

[Tools in AI Configs | LaunchDarkly | Documentation]

IMO this feel like they're trying to do too much. Trying to apply "dynamic configuration" to every aspect of AI workflows, including agents and tools. Not sure if they're well-positioned in the stack to dictate agentic workflows and tool orchestration, probably better suited for dedicated AI agent framework code/platform. But I like that Launch Darkly has a pulse on AI/agentic trends.


AI Configs / Veeam Intelligence V1

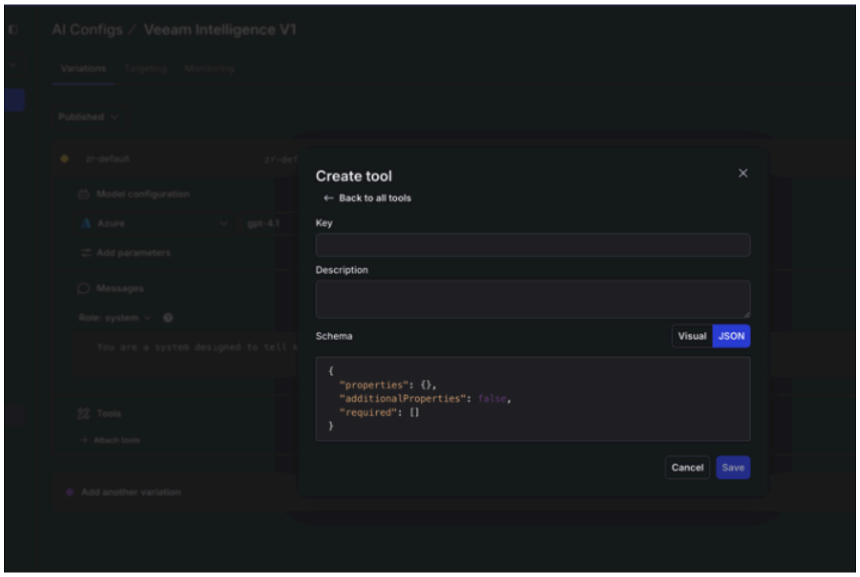
zr-default zr-default

Model configuration: Azure, gpt-4.1

Messages

Role: system

```
You are a system designed to tell knock-knock jokes.
```



```
You are a system designed to tell
```

```
{
  "properties": {},
  "additionalProperties": false,
  "required": []
}
```

TODO

Test dynamic prompts with placeholder values in the AI config prompt

POC Code

Here's the PR with the updates needed to integrate AI configs into Veeam Intelligence for VDC

https://github.com/veeam-ai/veeam-intelligence/pull/742 RESTRICTED CONTENT

Analysis

TODO Zack