Murakkab: Resource-Efficient Agentic Workflow Orchestration in Cloud Platforms

Gohar Irfan Chaudhry, Esha Choukse, Haoran Qiu, Íñigo Goiri,
Rodrigo Fonseca, Adam Belay, Ricardo Bianchini

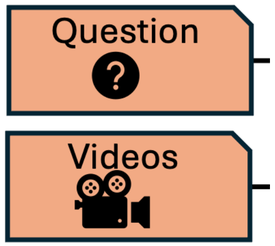
Agentic workflows are ubiquitous…




In this talk, we will focus on the resource-efficiency of agentic workflows

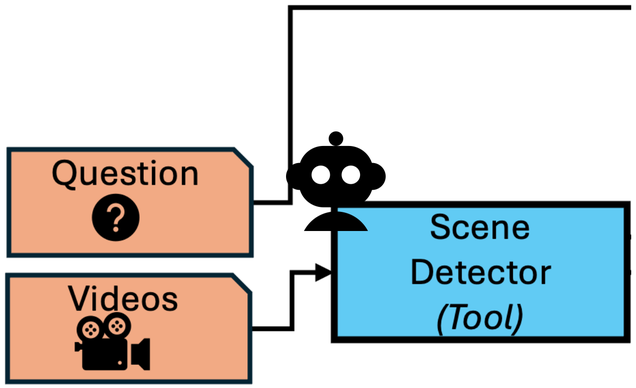
Example: Video Q/A workflow

Given a set of videos, answer the queries



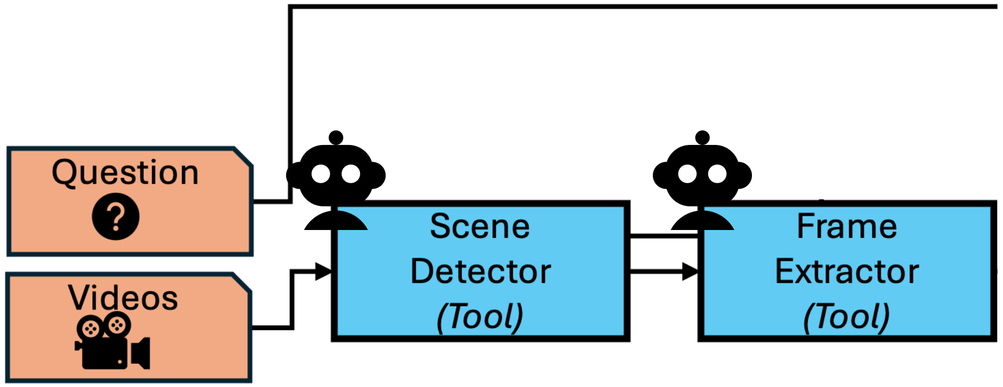
Example: Video Q/A workflow

Given a set of videos, answer the queries



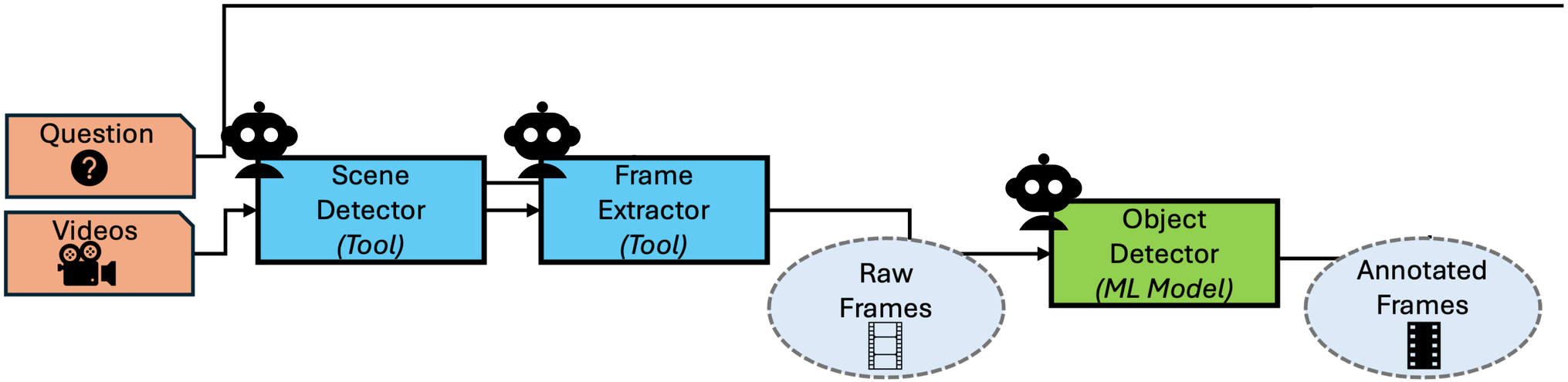
Example: Video Q/A workflow

Given a set of videos, answer the queries



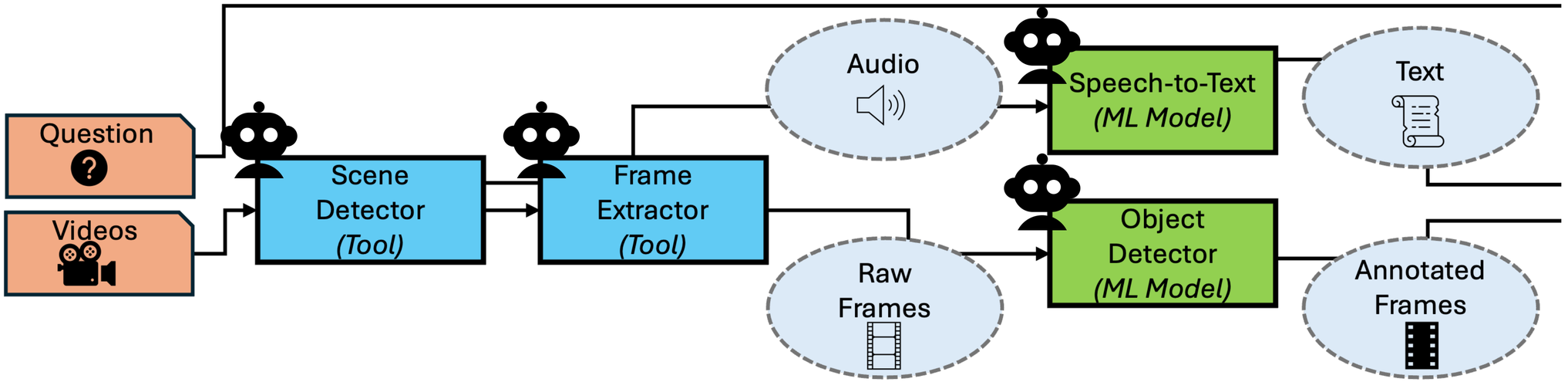
Example: Video Q/A workflow

Given a set of videos, answer the queries



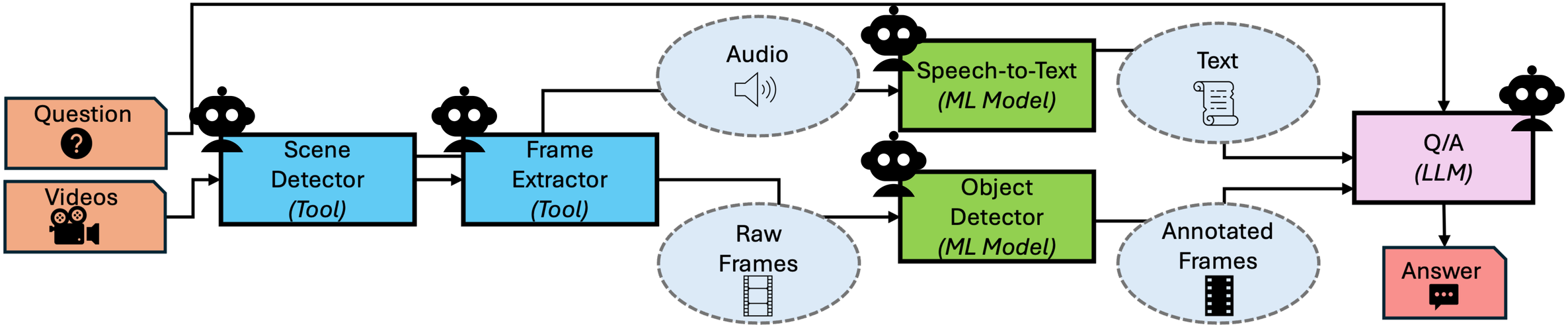
Example: Video Q/A workflow

Given a set of videos, answer the queries



Example: Video Q/A workflow

Given a set of videos, answer the queries

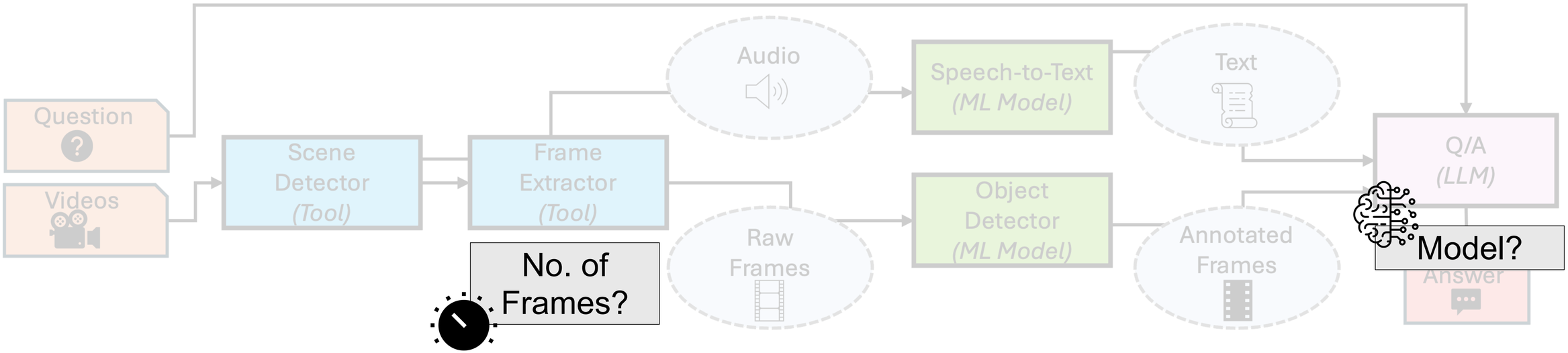


Example: Video Q/A workflow

Agent-, workflow-, hardware-level configurations

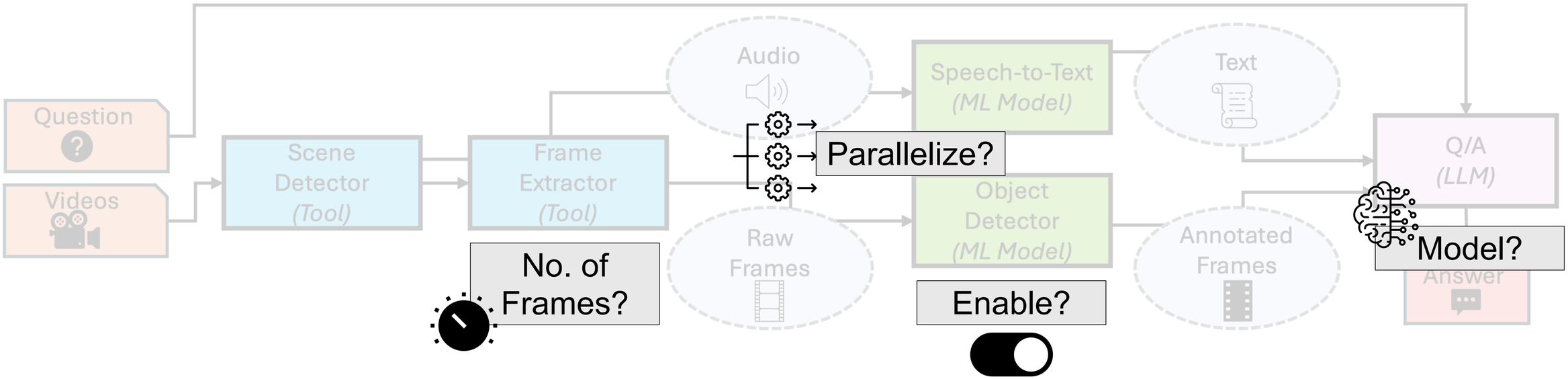
Example: Video Q/A workflow

Agent-, workflow-, and hardware-level configurations



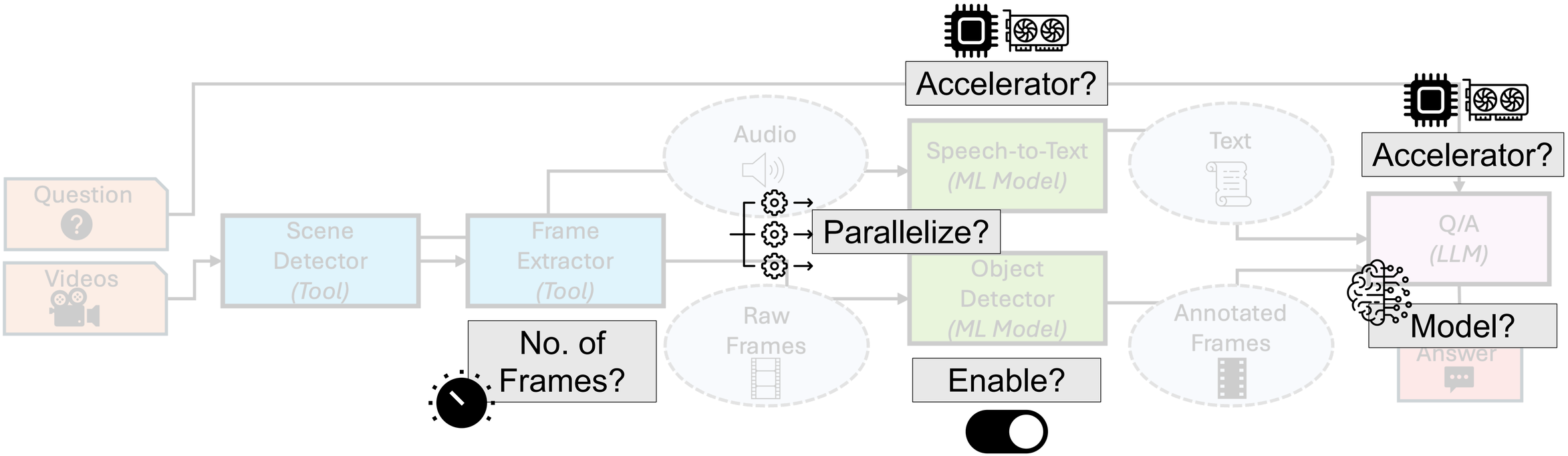
Example: Video Q/A workflow

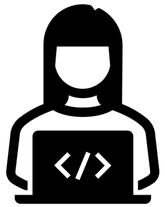
Agent-, workflow-, and hardware-level configurations



Example: Video Q/A workflow

Agent-, workflow-, and hardware-level configurations





App. Developer

How is a workflow defined today?

Example: Video Q/A workflow definition

```
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the
     developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the
     developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Example: Video Q/A workflow definition

```python
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the
     developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the
     developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Application Logic

Example: Video Q/A workflow definition

```python
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Application Logic

Example: Video Q/A workflow definition

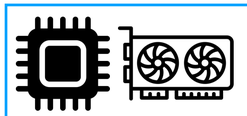
```python
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the
     developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the
     developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Application Logic

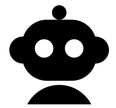
Example: Video Q/A workflow definition

```python
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Application Logic



Hardware Config.



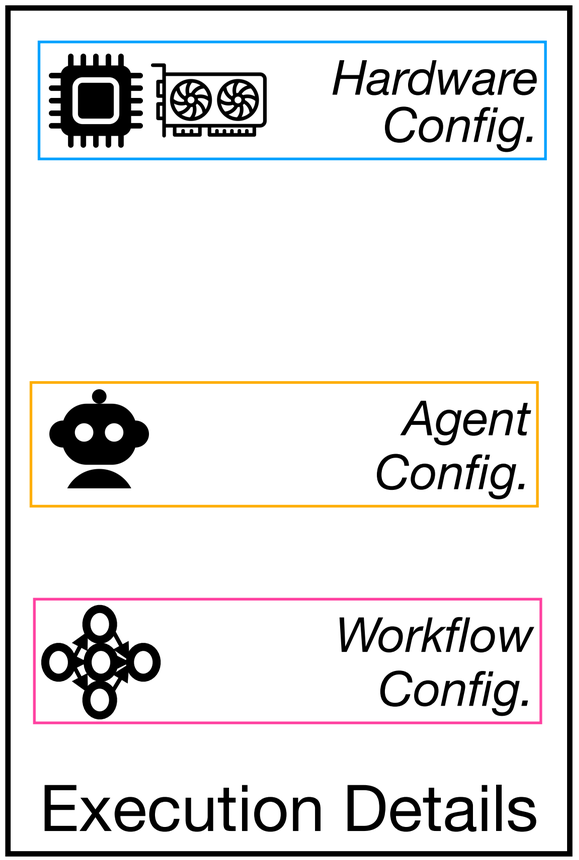
Agent Config.

Workflow Config.

Example: Video Q/A workflow definition

```
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the
      developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the
      developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Application Logic



Example: Video Q/A workflow definition

Tightly coupled application logic and execution details…

```python
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Application Logic

Hardware Config.

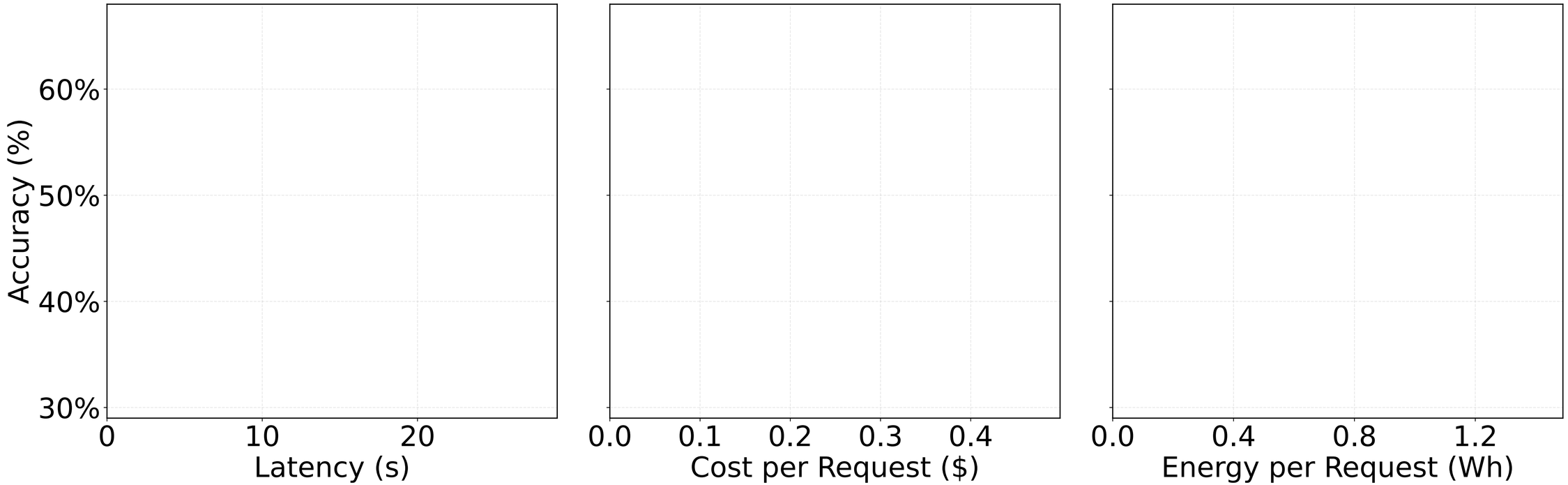
Agent Config.

Workflow Config.

Execution Details

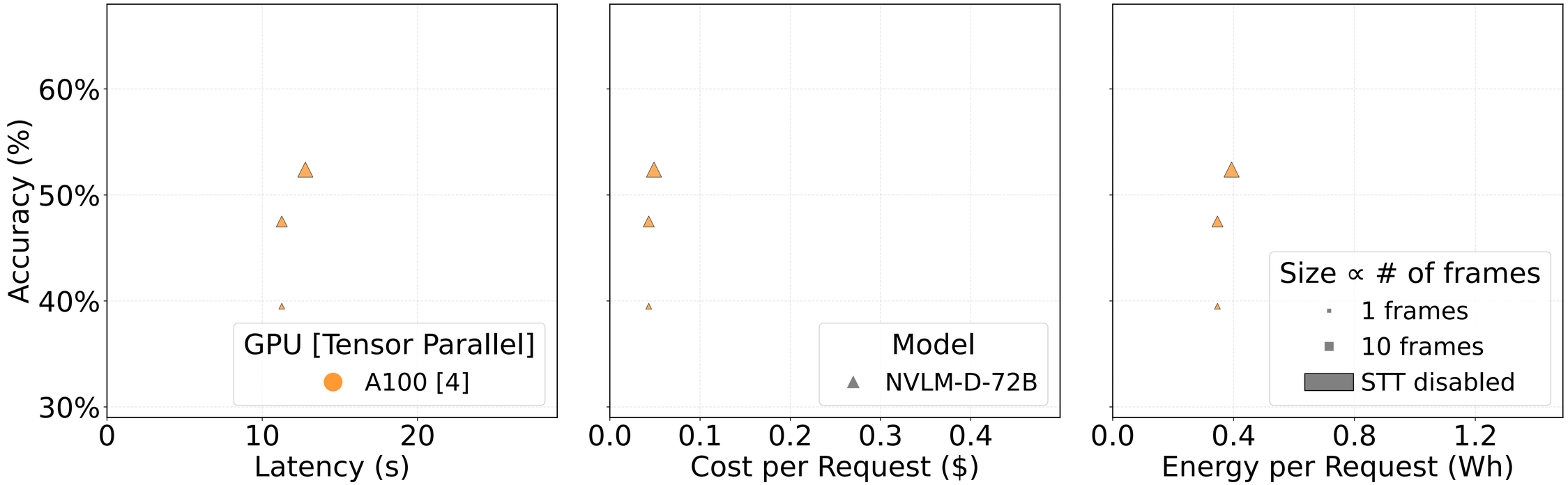
High-dimensional configuration space

Multiple objectives/SLOs



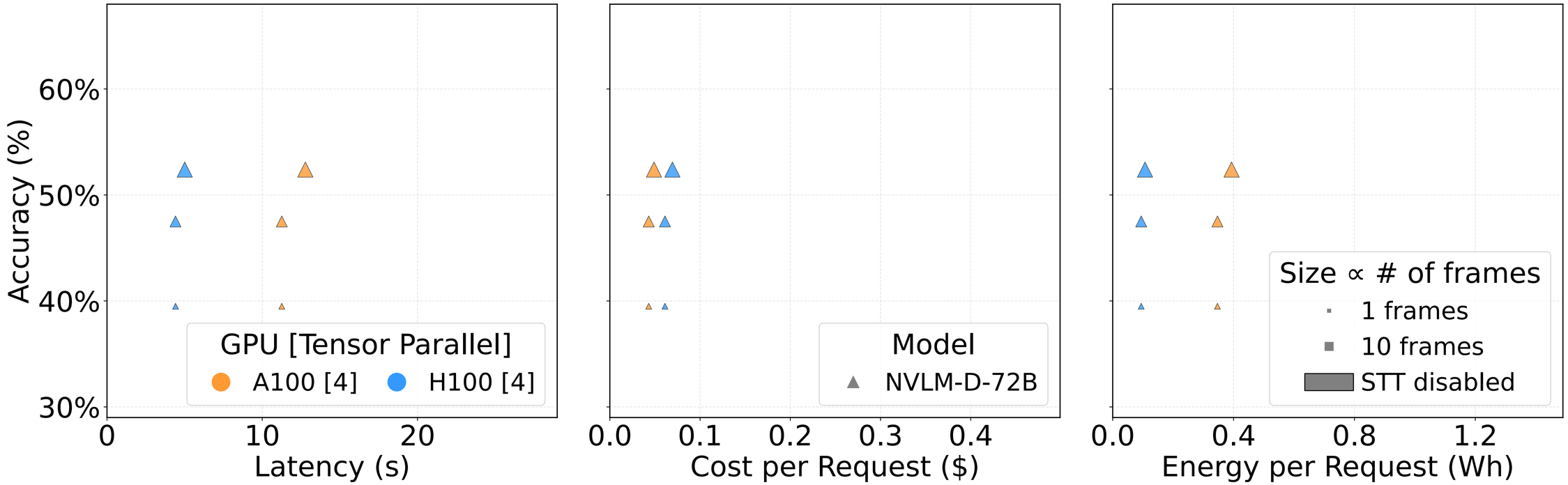
High-dimensional configuration space

Multiple objectives/SLOs



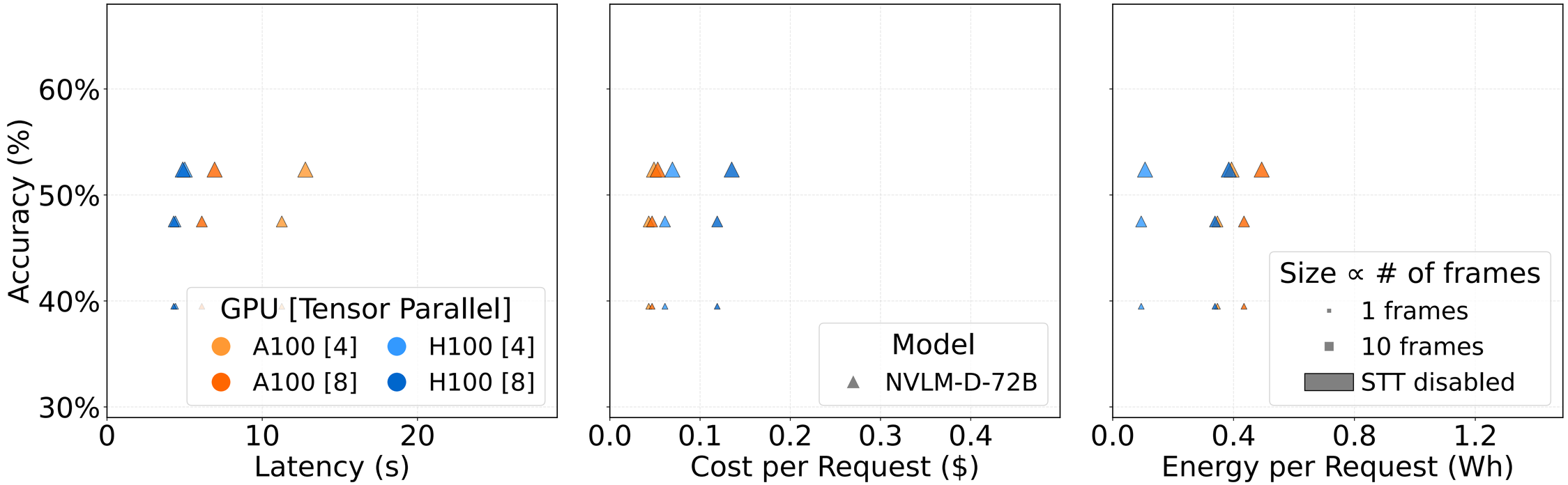
High-dimensional configuration space

Multiple objectives/SLOs



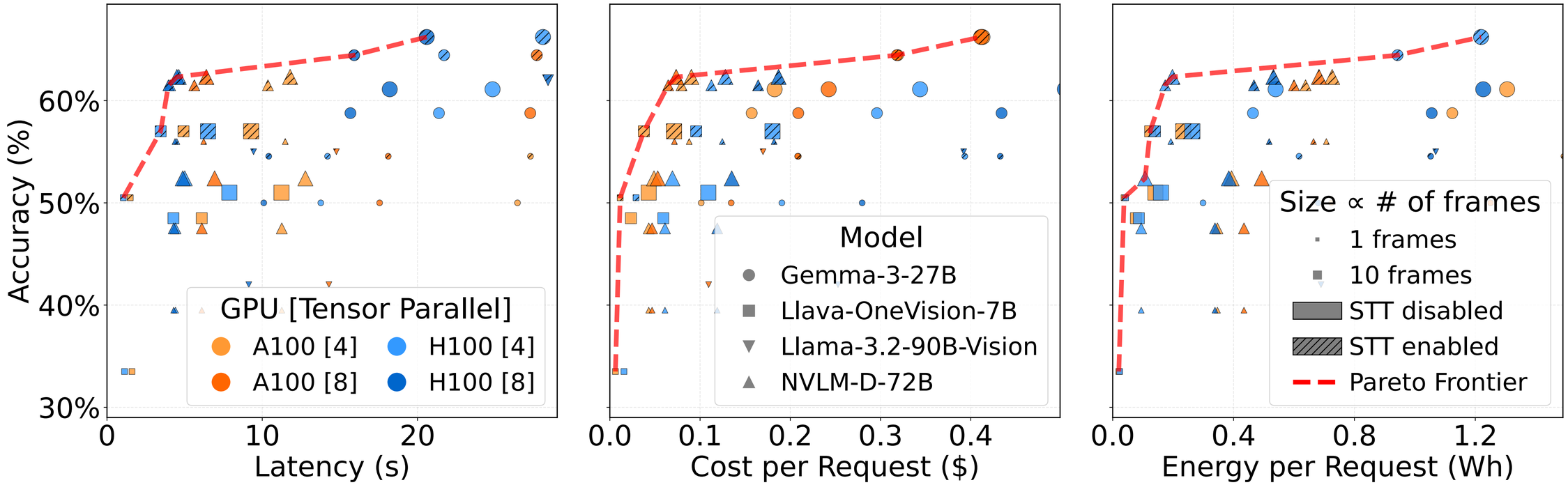
High-dimensional configuration space

Multiple objectives/SLOs



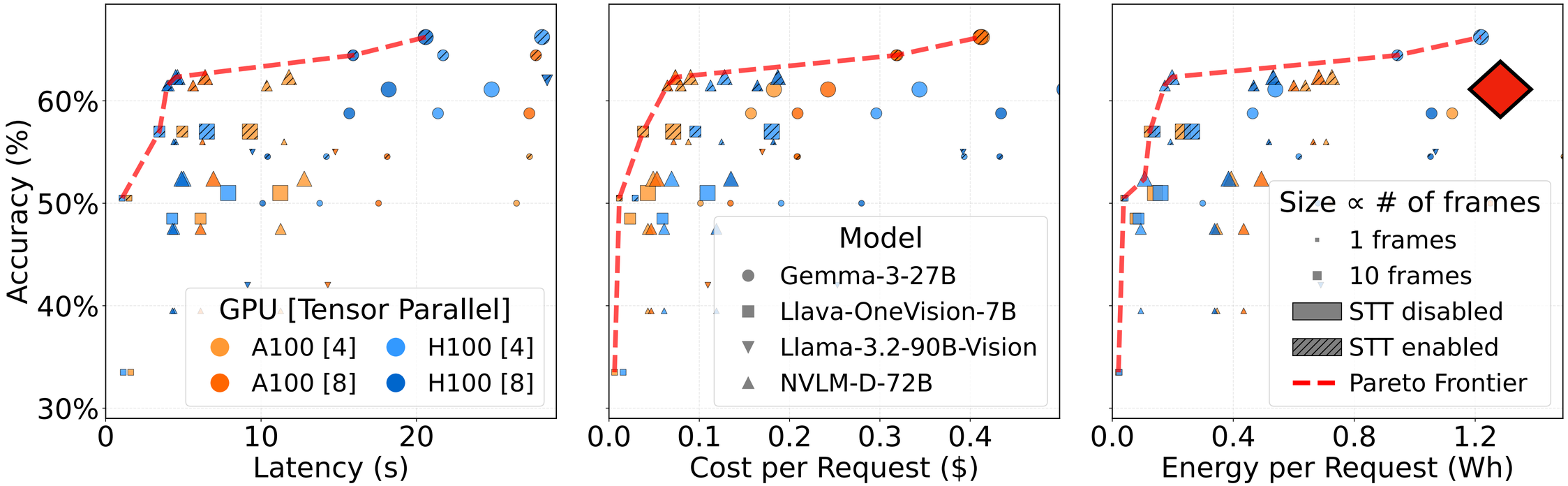
High-dimensional configuration space

Multiple objectives/SLOs



High-dimensional configuration space

Manual configuration is intractable and often suboptimal…



Sources of resource-inefficiency today:

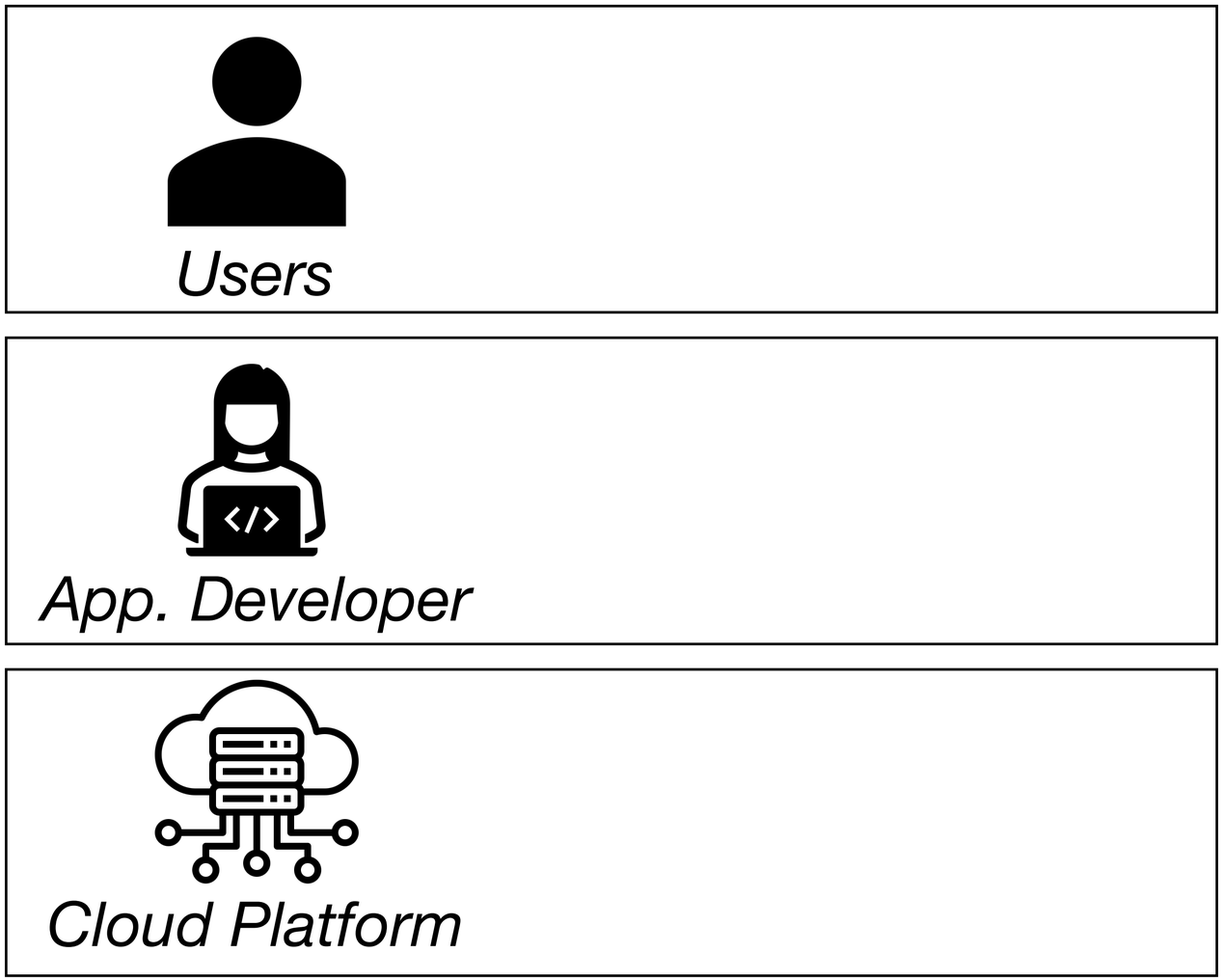
(1) Rigid workflows→ difficult to adapt

Entities involved in an agentic workflow’s lifecycle

Entities involved in an agentic workflow’s lifecycle

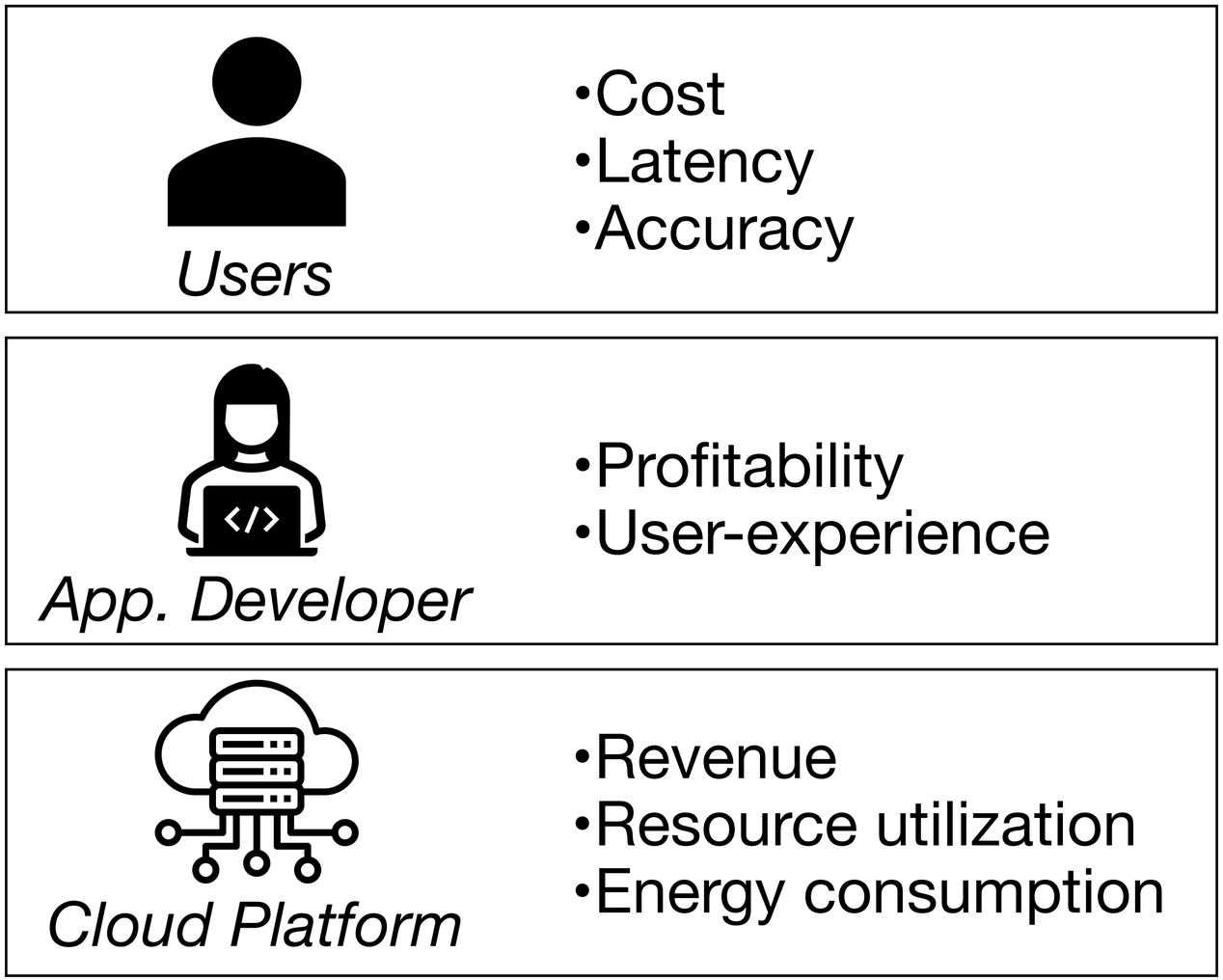
Entities involved in an agentic workflow’s lifecycle

Entities involved in an agentic workflow’s lifecycle



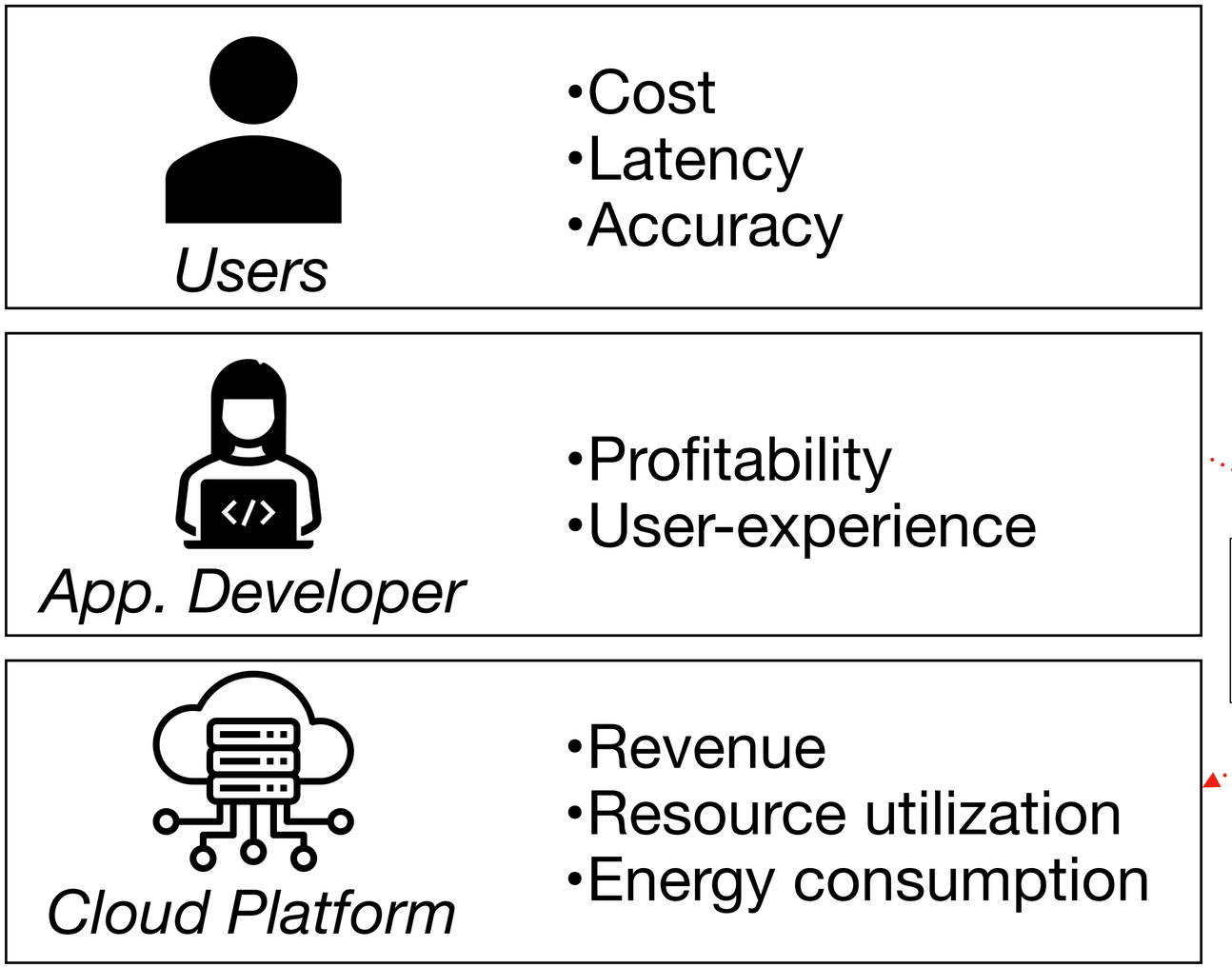
Entities involved in an agentic workflow’s lifecycle

Objectives for each entity in the stack



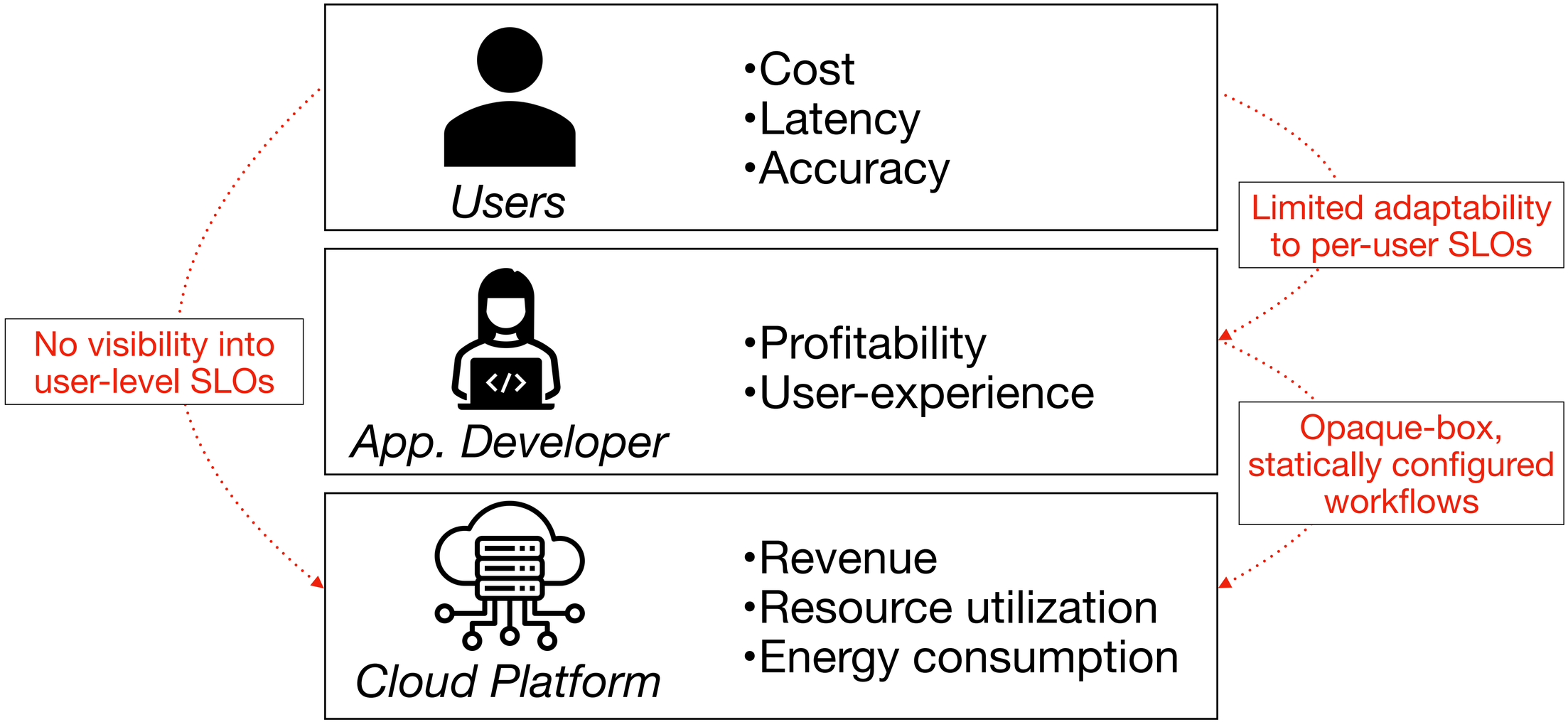
Entities involved in an agentic workflow's lifecycle

Exist in silos, limited cross-stack visibility…



Entities involved in an agentic workflow's lifecycle

Exist in silos, limited cross-stack visibility…



Sources of resource-inefficiency today:

(1) Rigid workflows→ difficult to adapt

(2) Fragmented stack→ limits cross-stack optimization

We need:

Unified, end-to-end orchestration to deliver high resource-efficiency

Murakkab

Resource-Efficient Agentic Workflow Orchestration in Cloud Platforms

Murakkab

Resource-Efficient Agentic Workflow Orchestration in Cloud Platforms

1. Declarative Workflow Specification

```
# == Sub-tasks in the workflow ==
scene_detect   = "Given a list of videos, identify scenes in each."
frame_extract  = "Given a list of scenes, extract frames."
stt            = "Given a list of scenes, convert audio to text."
q_a            = "Answer the query given some context."
# == Workflow description (sub-tasks and data flow) ==
def workflow(query, videos):
    scenes      = scene_detect(videos)
    frames      = frame_extract(scenes)
    transcript  = stt(scenes)
    answer      = q_a(query, [frames, transcript])
    return answer
# == Execution with example request ==
query   = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
result  = run(workflow(query, videos), slo=LOW_LATENCY)
```

Separate the “what” from the “how”.

Murakkab

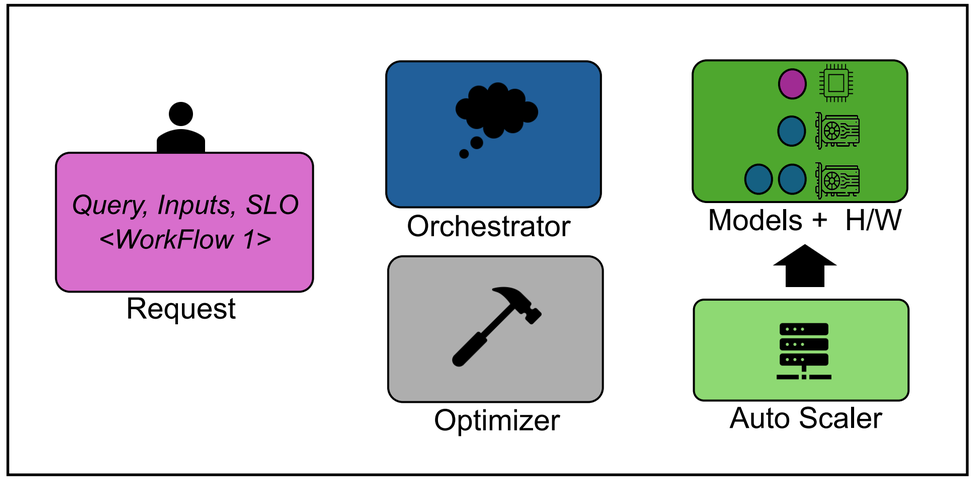
Resource-Efficient Agentic Workflow Orchestration in Cloud Platforms

1. Declarative Workflow Specification

```
# == Sub-tasks in the workflow ==
scene_detect    = "Given a list of videos, identify scenes in each."
frame_extract   = "Given a list of scenes, extract frames."
stt             = "Given a list of scenes, convert audio to text."
q_a             = "Answer the query given some context."
# == Workflow description (sub-tasks and data flow) ==
def workflow(query, videos):
    scenes      = scene_detect(videos)
    frames      = frame_extract(scenes)
    transcript  = stt(scenes)
    answer      = q_a(query, [frames, transcript])
    return answer
# == Execution with example request ==
query   = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
result  = run(workflow(query, videos), slo=LOW_LATENCY)
```

Separate the "what" from the "how".

2. Adaptive, SLO-aware Orchestration

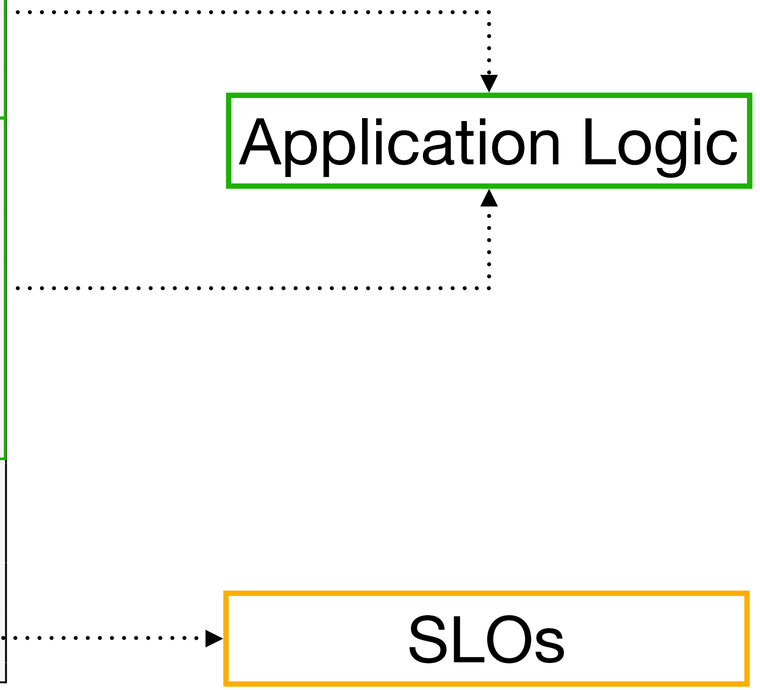


Configuration is a runtime decision, not a development decision.

Phase 1: Development

Declarative Workflow Specification

```
# == Sub-tasks in the workflow ==
scene_detect   = "Given a list of videos, identify scenes in each."
frame_extract  = "Given a list of scenes, extract frames."
stt            = "Given a list of scenes, convert audio to text."
q_a            = "Answer the query given some context."
# == Workflow description (sub-tasks and data flow) ==
def workflow(query, videos):
    scenes      = scene_detect(videos)
    frames      = frame_extract(scenes)
    transcript  = stt(scenes)
    answer      = q_a(query, [frames, transcript])
    return answer
# == Execution with example request ==
query   = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
result  = run(workflow(query, videos), slo=LOW_LATENCY)
```



Challenge

How do we capture the missing details?

```python
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

?

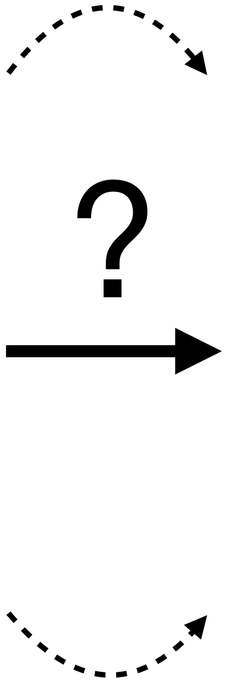
```python
# == Sub-tasks in the workflow ==
scene_detect   = "Given a list of videos, identify scenes in each."
frame_extract  = "Given a list of scenes, extract frames."
stt            = "Given a list of scenes, convert audio to text."
q_a            = "Answer the query given some context."
# == Workflow description (sub-tasks and data flow) ==
def workflow(query, videos):
    scenes      = scene_detect(videos)
    frames      = frame_extract(scenes)
    transcript  = stt(scenes)
    answer      = q_a(query, [frames, transcript])
    return answer
# == Execution with example request ==
query   = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
result  = run(workflow(query, videos), slo=LOW_LATENCY)
```

Challenge

How do we capture the missing details?

```python
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Which agents to use?




```python
# == Sub-tasks in the workflow ==
scene_detect   = "Given a list of videos, identify scenes in each."
frame_extract  = "Given a list of scenes, extract frames."
stt            = "Given a list of scenes, convert audio to text."
q_a            = "Answer the query given some context."
# == Workflow description (sub-tasks and data flow) ==
def workflow(query, videos):
    scenes      = scene_detect(videos)
    frames      = frame_extract(scenes)
    transcript  = stt(scenes)
    answer      = q_a(query, [frames, transcript])
    return answer
# == Execution with example request ==
query   = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
result  = run(workflow(query, videos), slo=LOW_LATENCY)
```

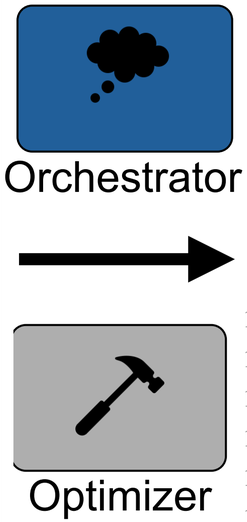
How to configure the workflow?

Challenge

Murakkab: Orchestrator + Optimizer

```
# == Workflow nodes (coupled app logic + configs) ==
scene_detection = Tool(
    fn=SceneDetector(),    # Implemented earlier by the
      developer.
    key=AWS_KEY, resources={"CPUs": 32}
)
frame_extractor = Tool(
    fn=FrameExtractor(),   # Implemented earlier by the
      developer.
    key=AWS_KEY, params={"num_frames": 15}, resources={"CPUs": 32}
)
speech_to_text = MLModel(
    name="Whisper", key=OPENAI_API_KEY, resources={"PTUs": 50}
)
object_detection = MLModel(
    name="CLIP", key=AZURE_KEY, resources={"CPUs": 128}
)
question_answer = LLM(
    name="Llama-3.2", key=DATABRICKS_API_KEY,
    params={"batch": 256}, resources={"GPUs": 8, "Type": "H100"},
    system_prompt="You are an agent that can understand videos.",
    user_prompt="Answer the given question about the video."
)
# == Query and data ==
ques    = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
# == Workflow (ordering of nodes and data flow) ==
scenes, audio  = scene_detection(videos)
frames         = frame_extractor(scenes)
transcript     = speech_to_text(audio)
obj_frames     = object_detection(frames)
answer         = question_answer(ques, transcript, obj_frames)
```

Which agents to use?

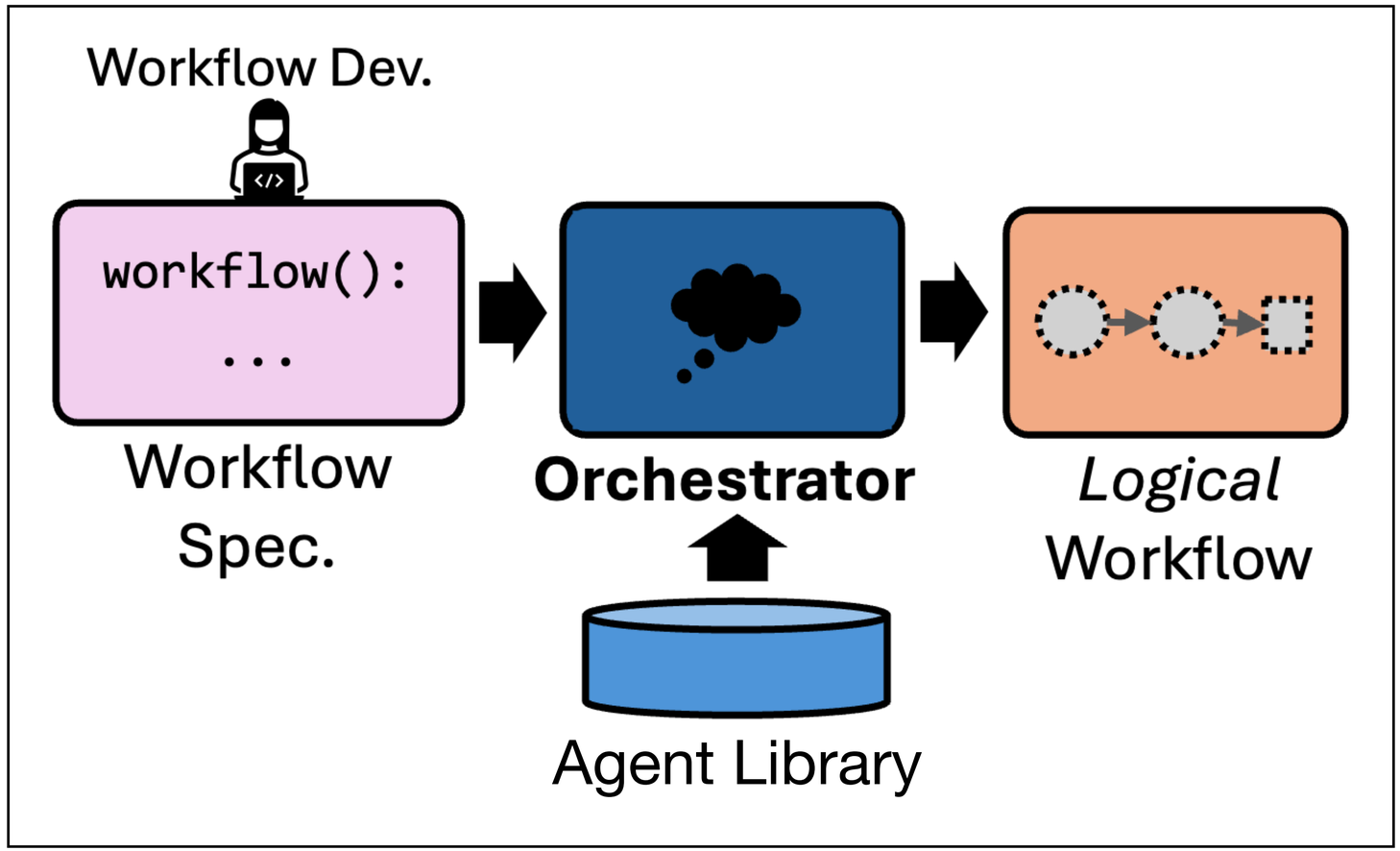



```
# == Sub-tasks in the workflow ==
scene_detect   = "Given a list of videos, identify scenes in each."
frame_extract  = "Given a list of scenes, extract frames."
stt            = "Given a list of scenes, convert audio to text."
q_a            = "Answer the query given some context."
# == Workflow description (sub-tasks and data flow) ==
def workflow(query, videos):
    scenes      = scene_detect(videos)
    frames      = frame_extract(scenes)
    transcript  = stt(scenes)
    answer      = q_a(query, [frames, transcript])
    return answer
# == Execution with example request ==
query   = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
result  = run(workflow(query, videos), slo=LOW_LATENCY)
```

How to configure the workflow?

Phase 1: Development

Orchestrator: Workflow Specification to Logical Workflow



Phase 1: Development

Separate the "what" from the "how"

```
# == Sub-tasks in the workflow ==
scene_detect   = "Given a list of videos, identify scenes in each."
frame_extract  = "Given a list of scenes, extract frames."
stt            = "Given a list of scenes, convert audio to text."
q_a            = "Answer the query given some context."
# == Workflow description (sub-tasks and data flow) ==
def workflow(query, videos):
    scenes      = scene_detect(videos)
    frames      = frame_extract(scenes)
    transcript  = stt(scenes)
    answer      = q_a(query, [frames, transcript])
    return answer
# == Execution with example request ==
query   = "What is the name of the person wearing the red dress?"
videos  = ["road_trip.mp4"]
result  = run(workflow(query, videos), slo=LOW_LATENCY)
```

FrameExtractor
(videos, <num_frames>)

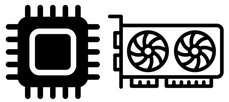
QuestionAnswer
(query, ctx, <model>)

Model
(<gpu_type>, <TP>)

Phase 2: Optimization + Execution

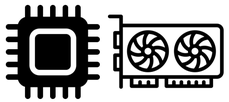
Profile-Guided Optimization

- **Workflow-level** profiles
- **Agent-level** profiles
- **Hardware-level** profiles



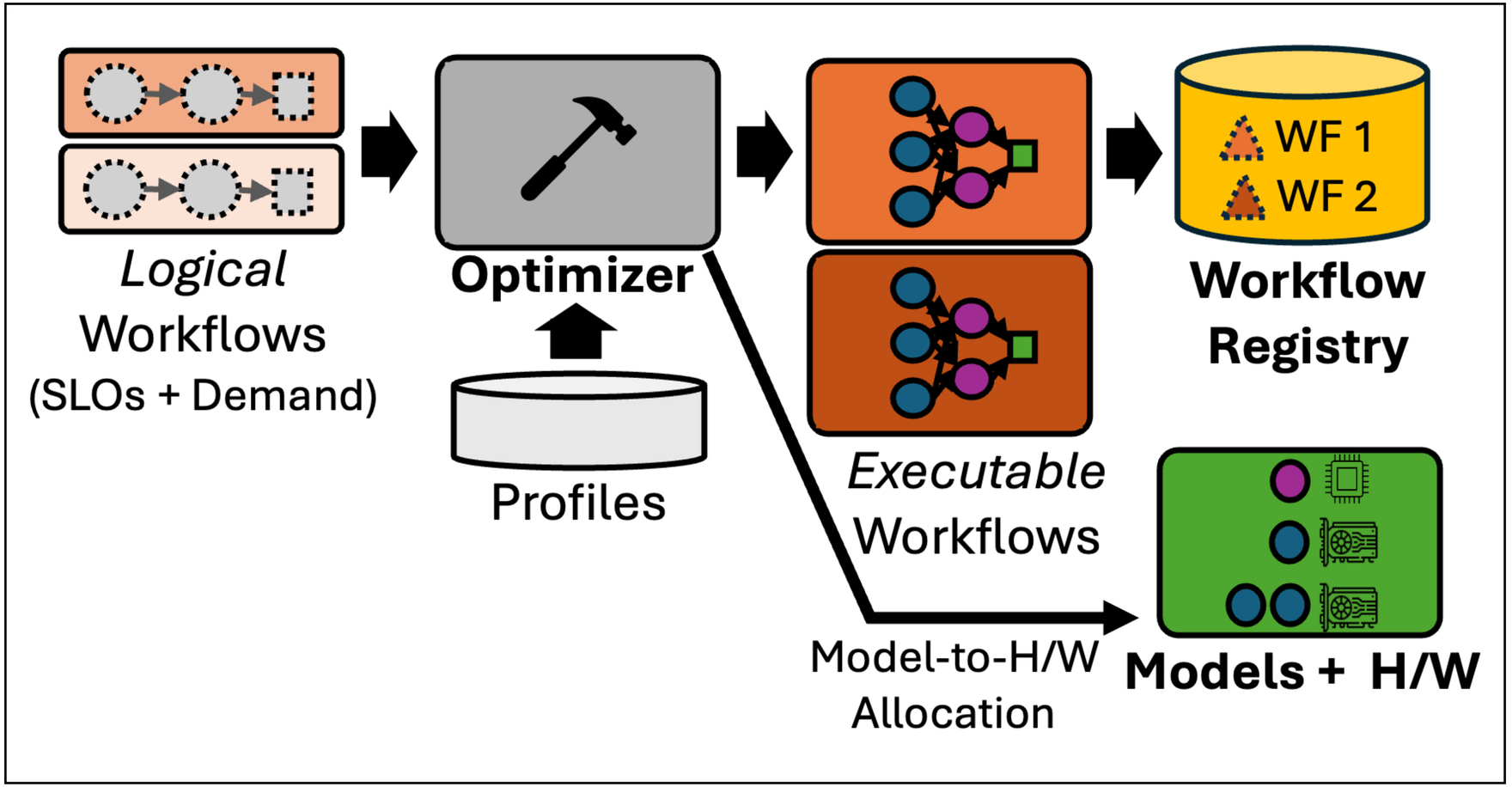
Phase 2: Optimization + Execution

Profile-Guided Optimization

- **Workflow-level** profiles 
- **Agent-level** profiles 
- **Hardware-level** profiles 
- Composable profiles across levels:
  - Make runtime optimization decisions (e.g., quality, latency, cost, energy)
  - Maximizes **reuse** across workflows
  - Minimizes re-profiling

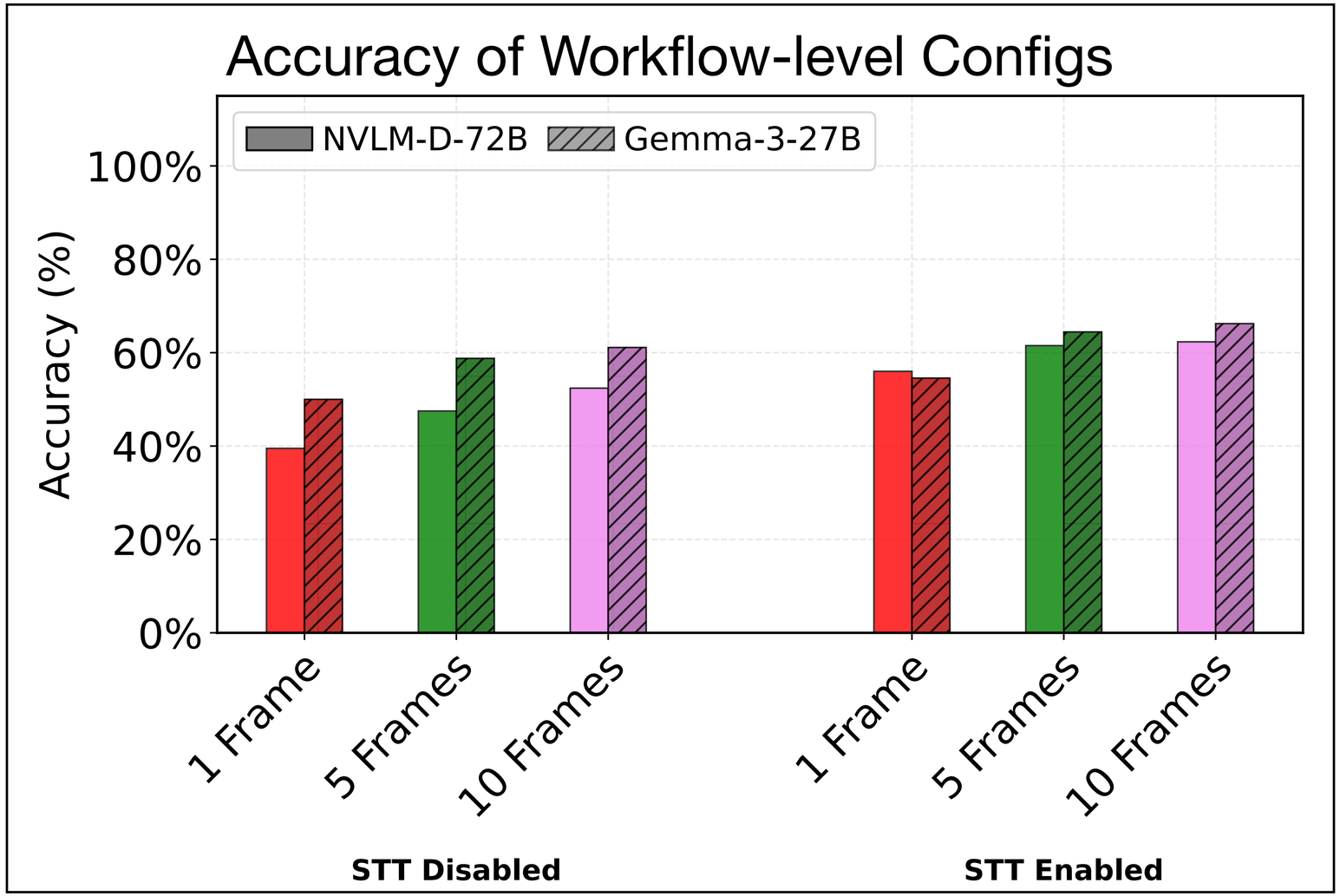
Phase 2: Optimization + Execution

Optimizer: Logical Workflow to Executable Workflow



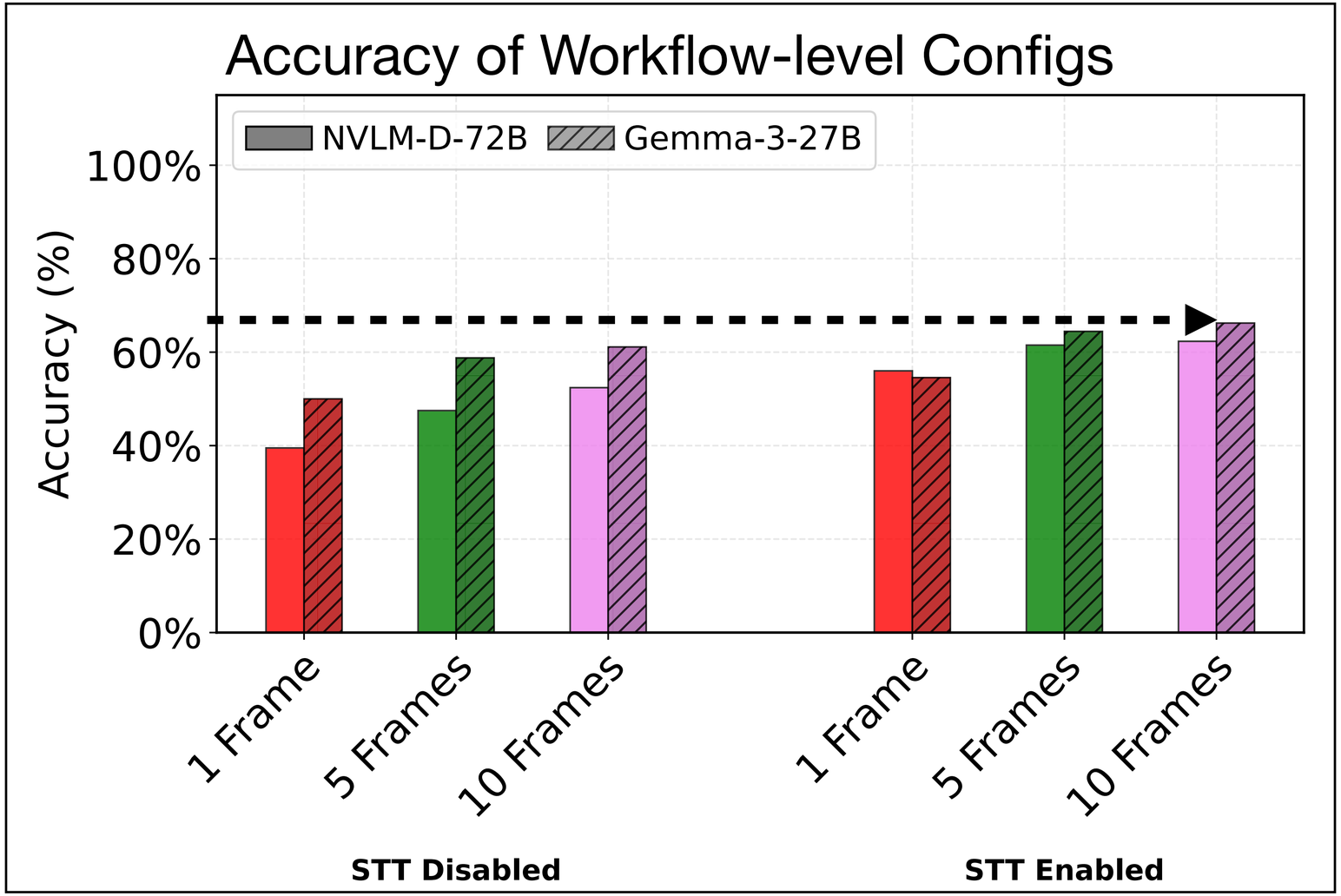
Phase 2: Optimization + Execution

Quality is determined using **workflow-level profiles**



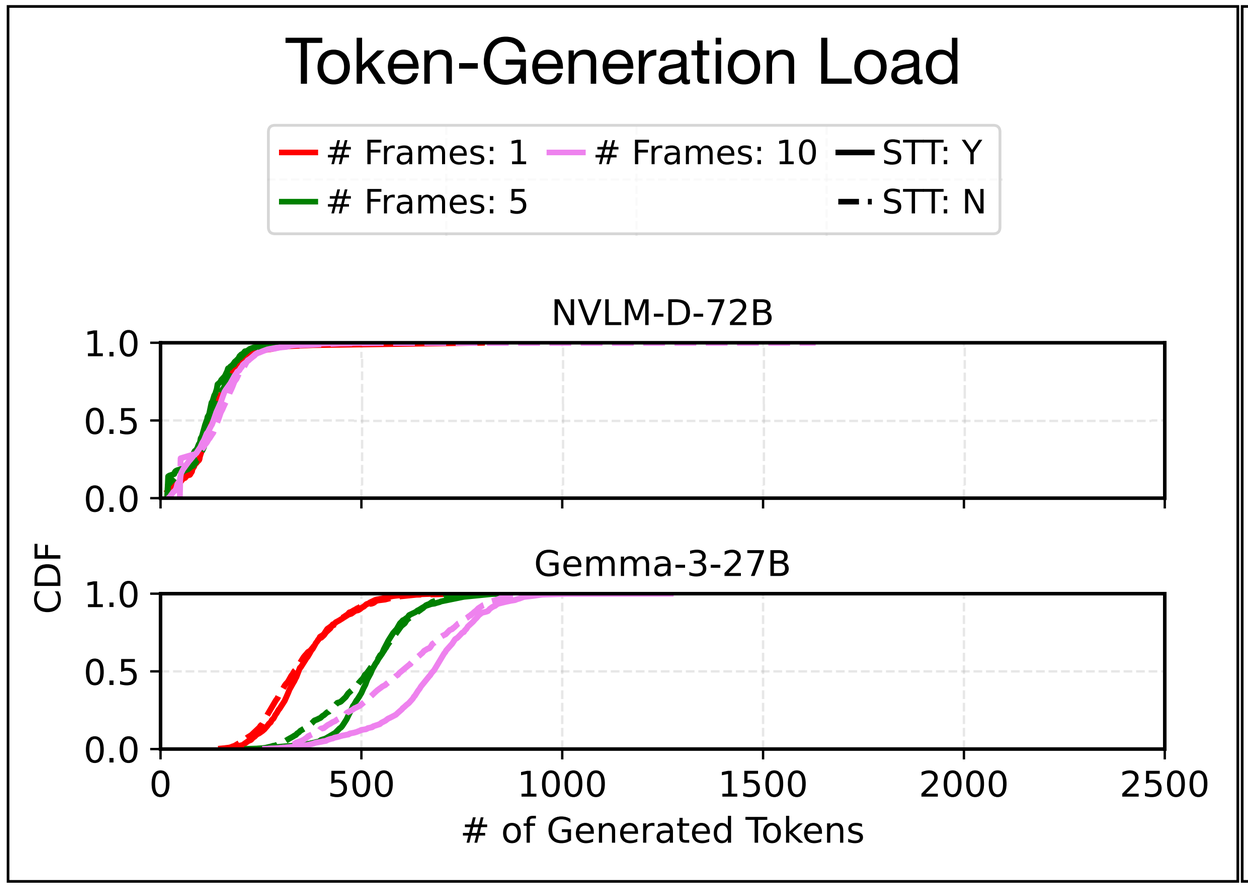
Phase 2: Optimization + Execution

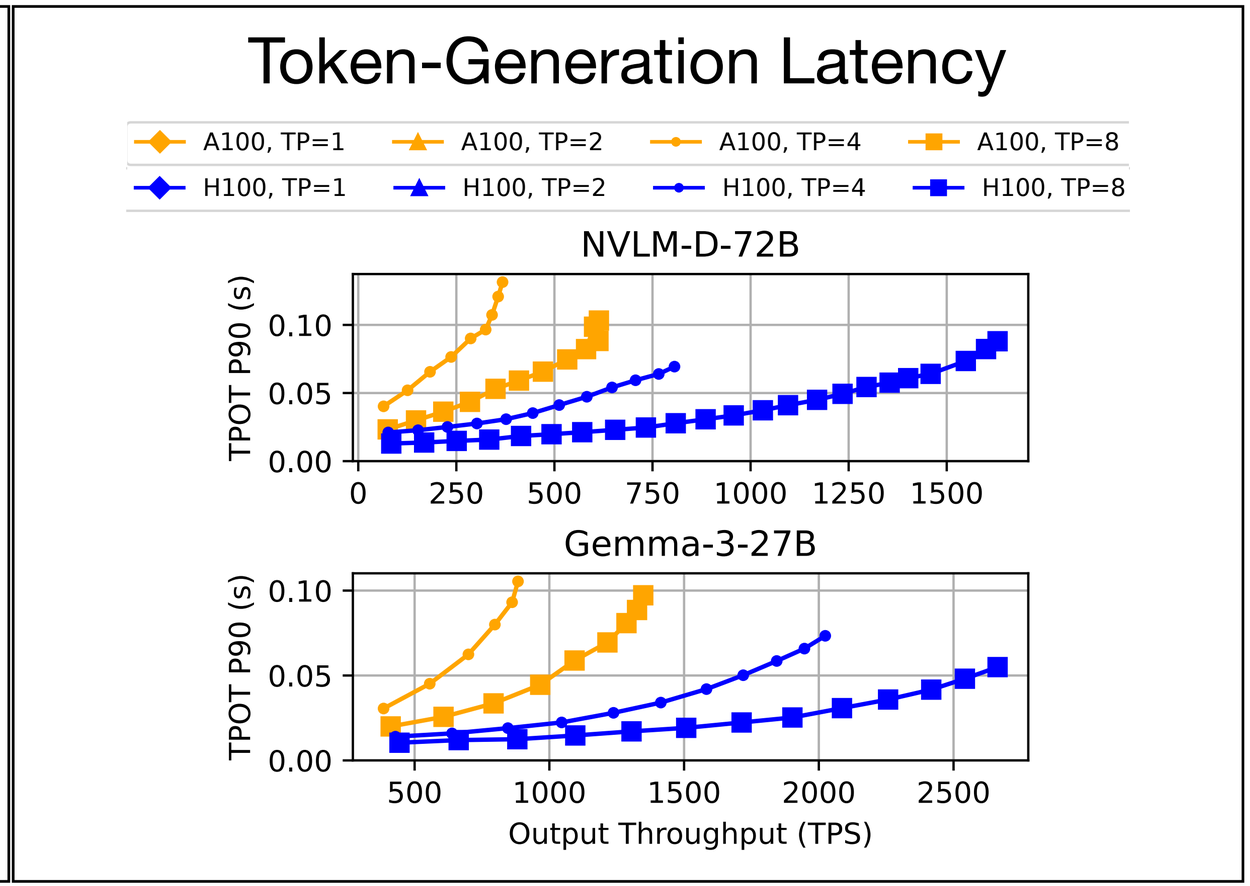
Quality is determined using **workflow-level profiles**



Phase 2: Optimization + Execution

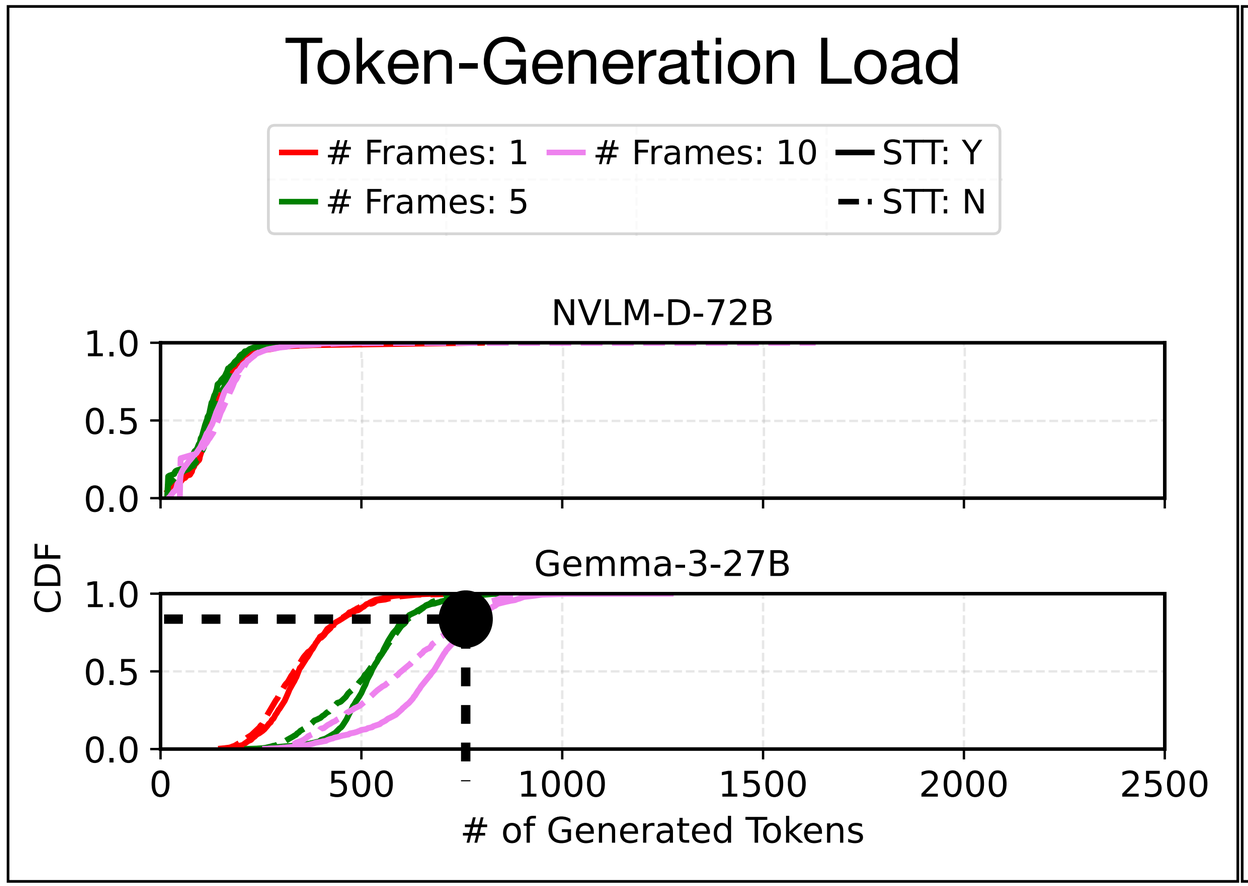
Latency/cost is determined by **agent- and hardware-level profiles**

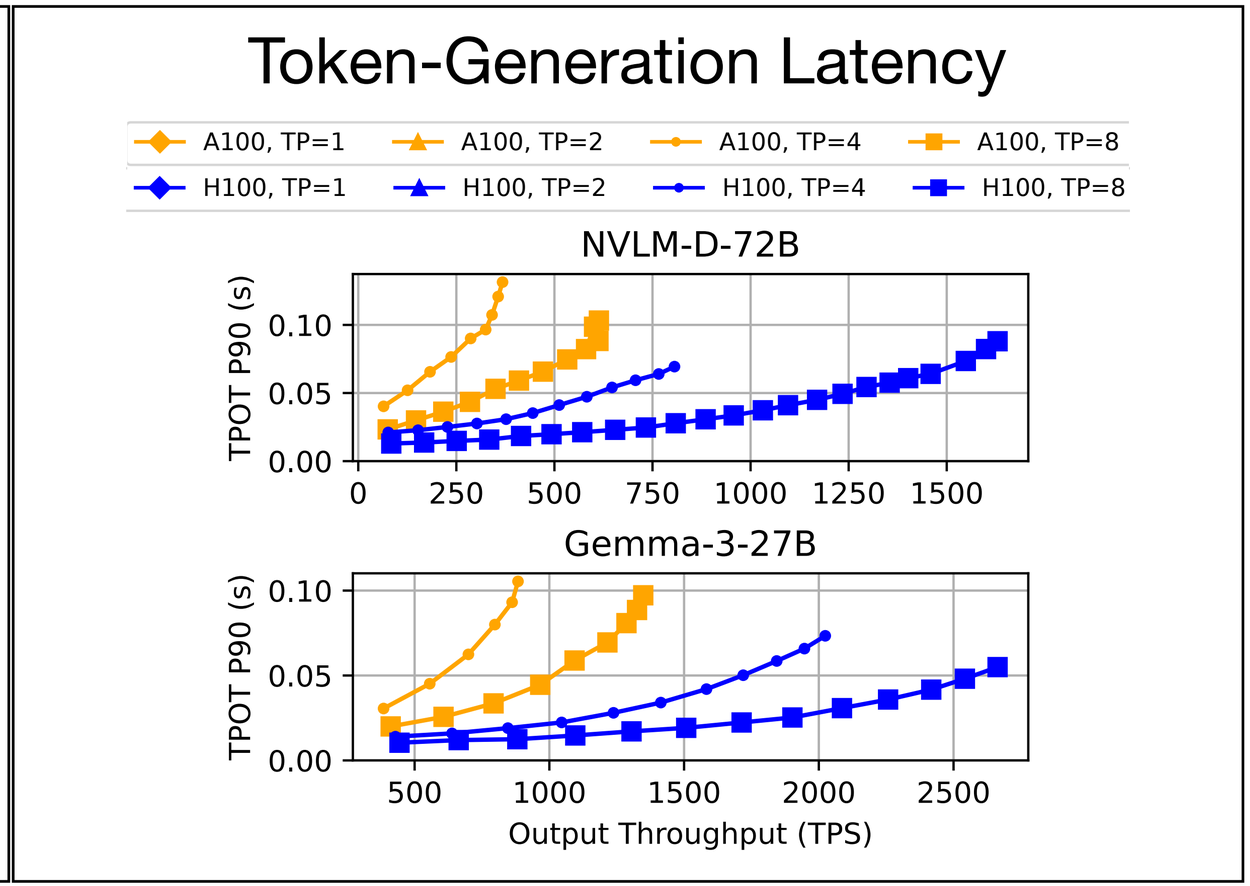




Phase 2: Optimization + Execution

Latency/cost is determined by **agent- and hardware-level profiles**

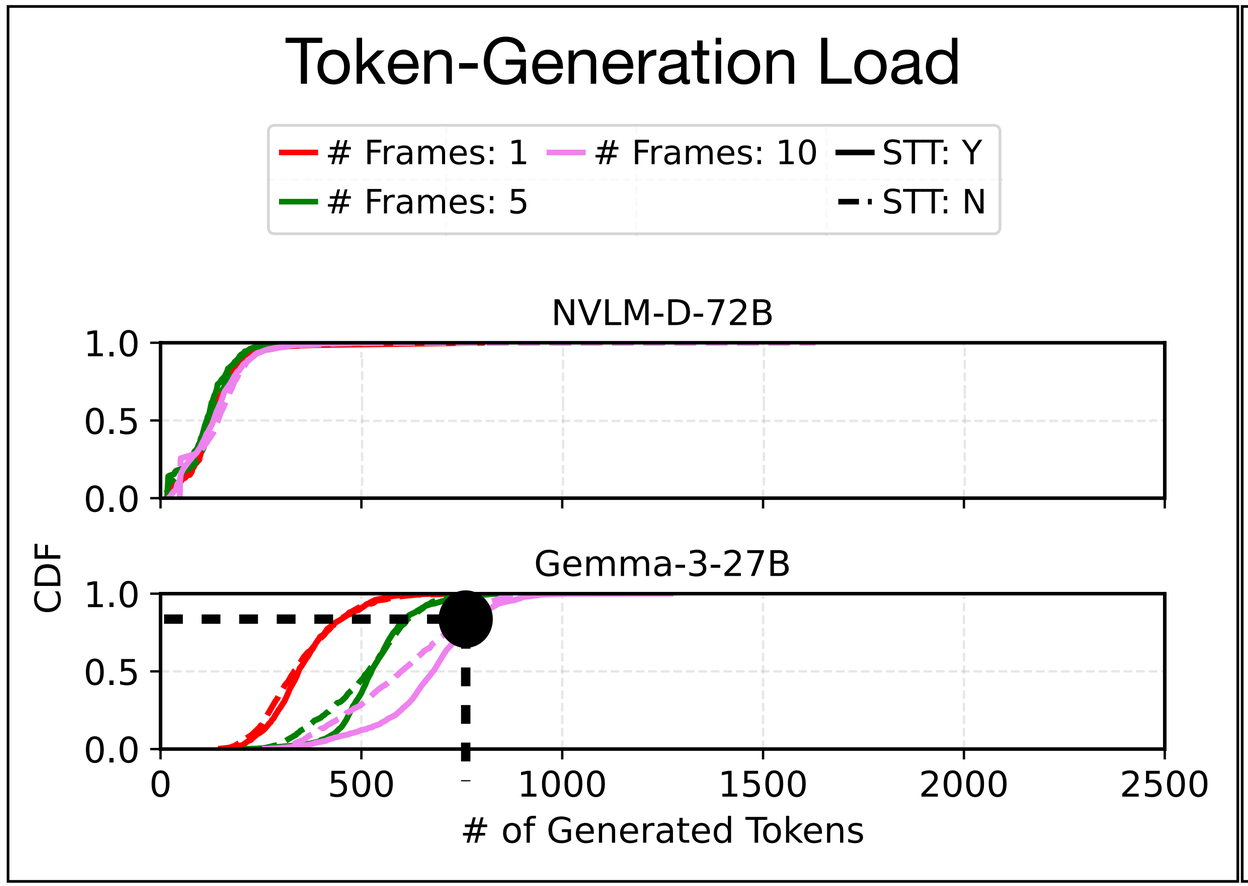


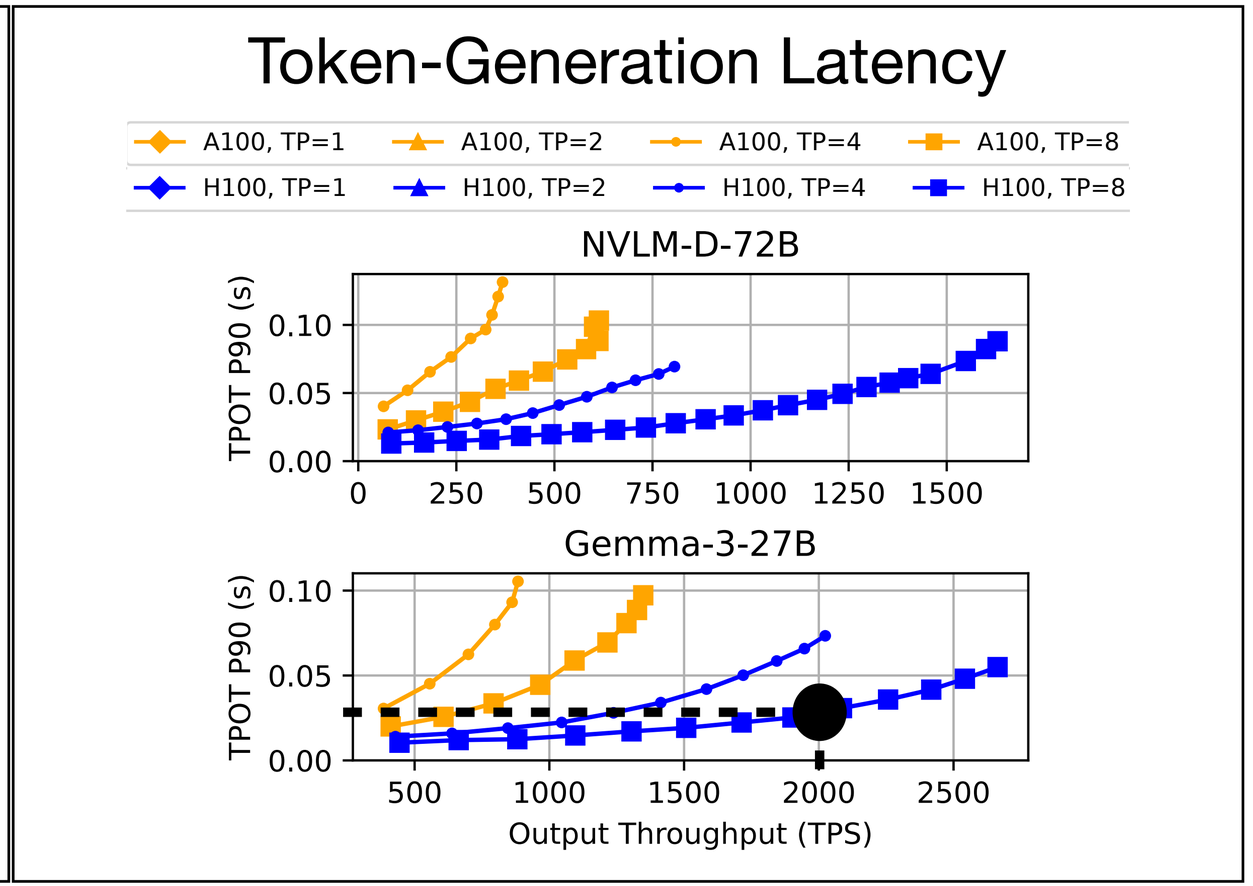


~700 tokens @ P90

Phase 2: Optimization + Execution

Latency/cost is determined by **agent- and hardware-level profiles**



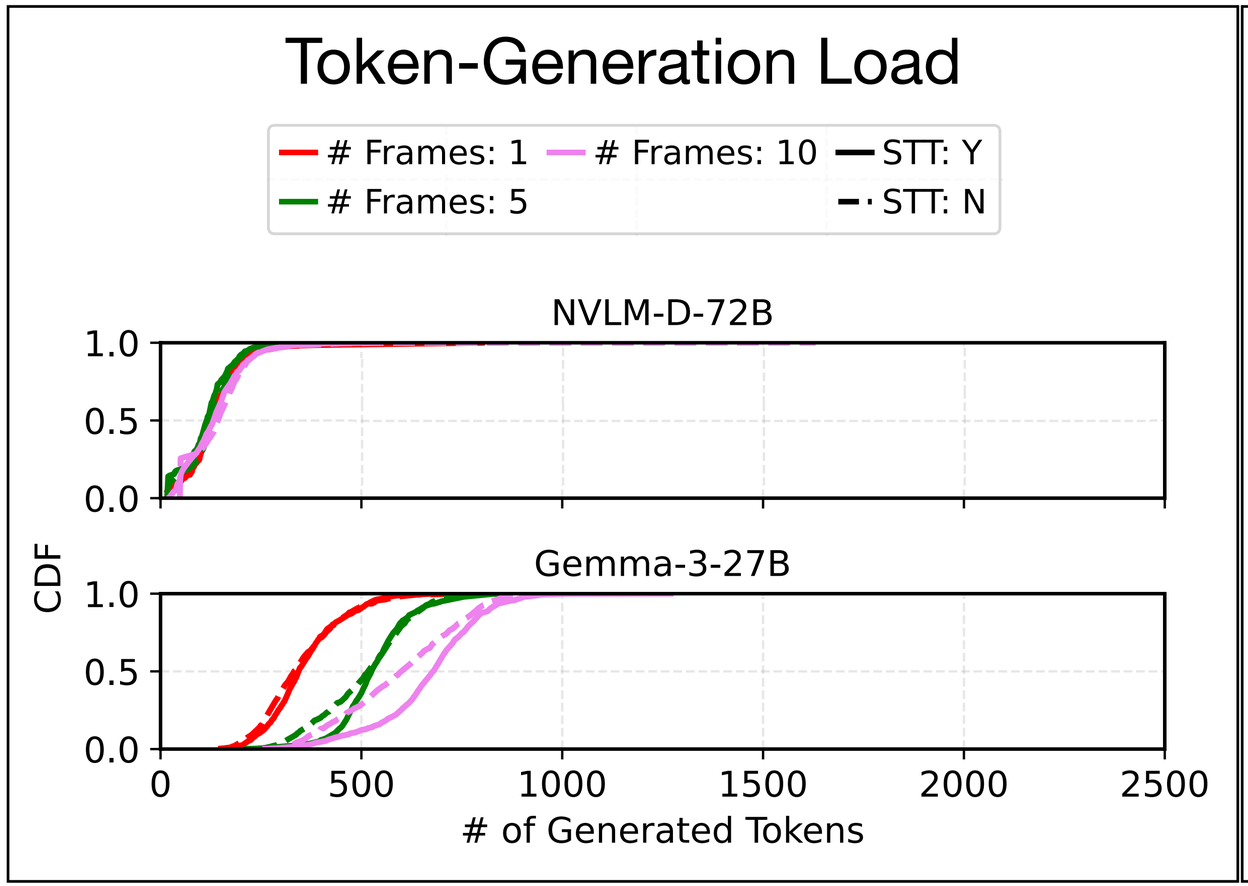


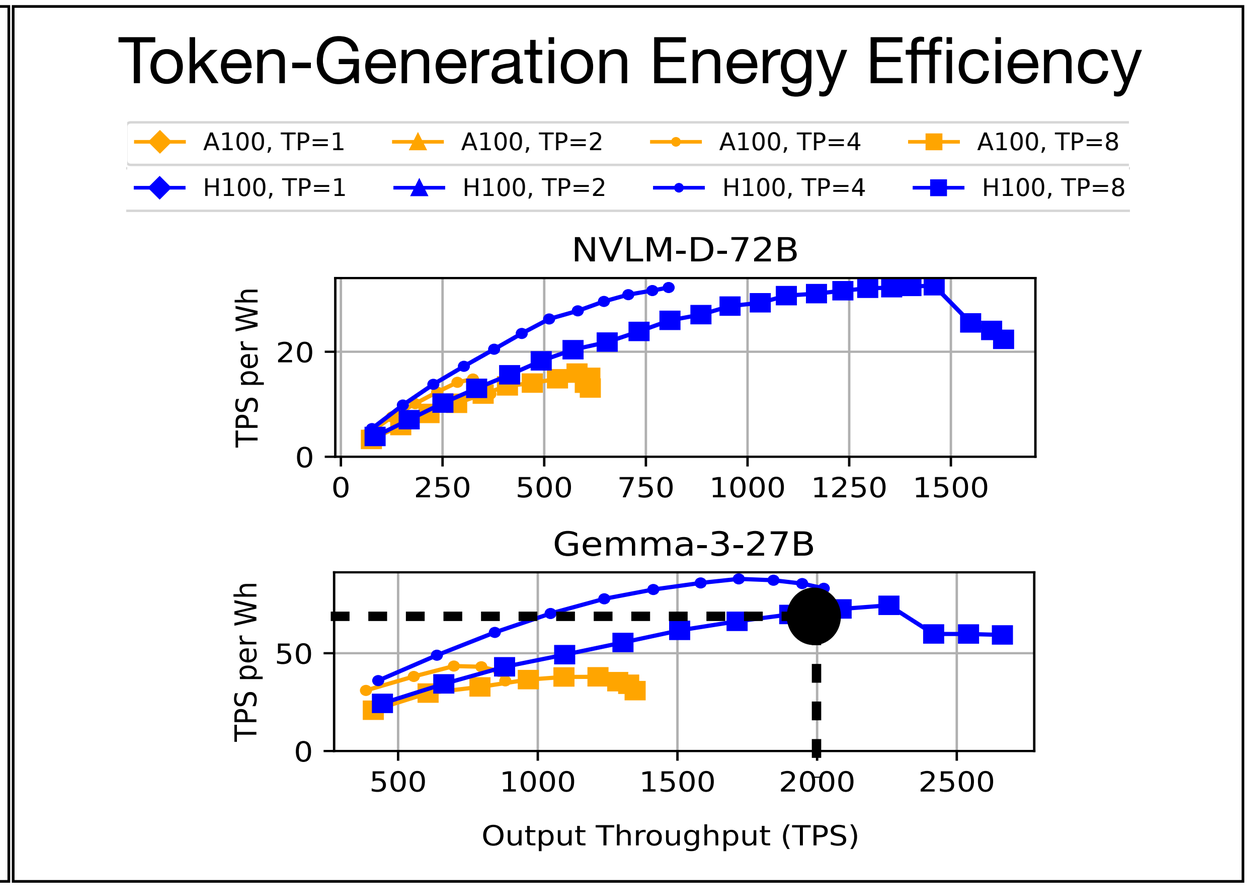
~700 tokens @ P90

~0.03s @ 2000 TPS — 700 x 0.03s = 21s

Phase 2: Optimization + Execution

Energy efficiency is determined by **agent- and hardware-level profiles**





Phase 2: Optimization + Execution

Configuration is a runtime decision, not a development decision

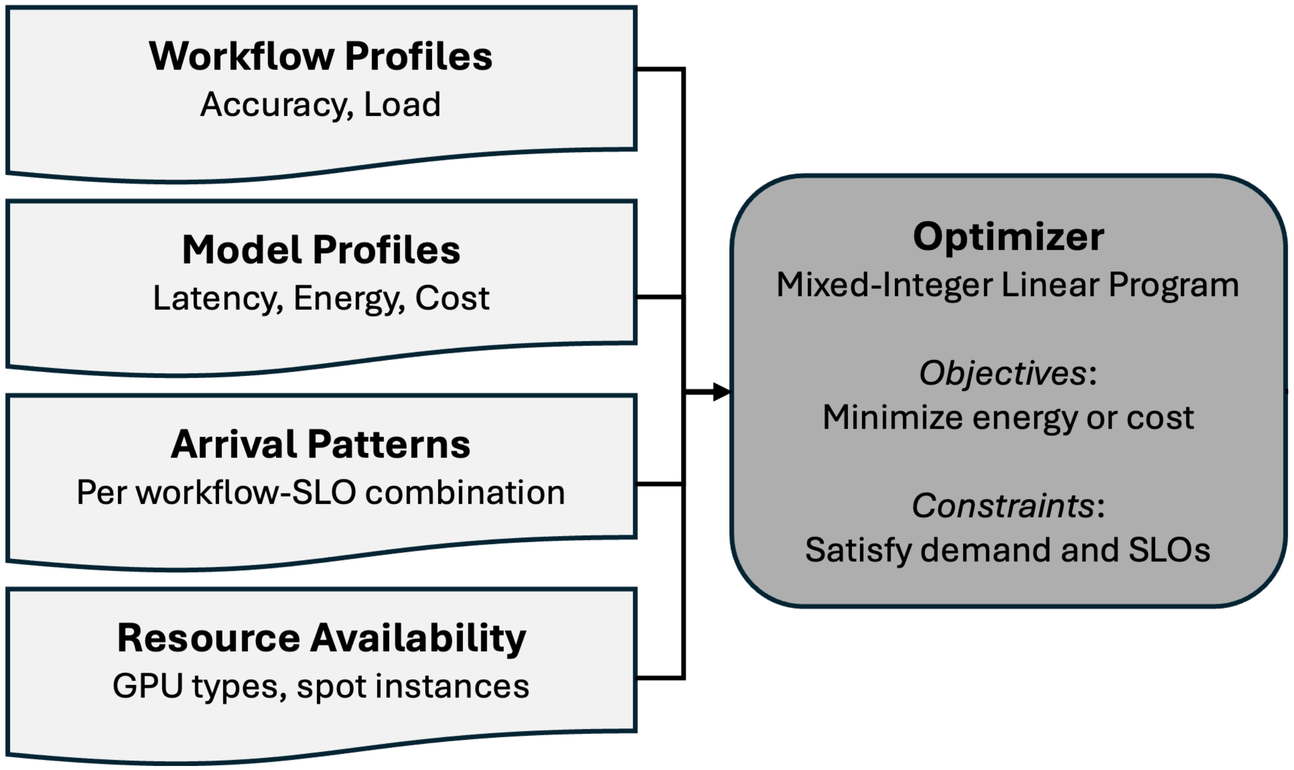
Optimizer
Mixed-Integer Linear Program

Objectives:
Minimize energy or cost

Constraints:
Satisfy demand and SLOs

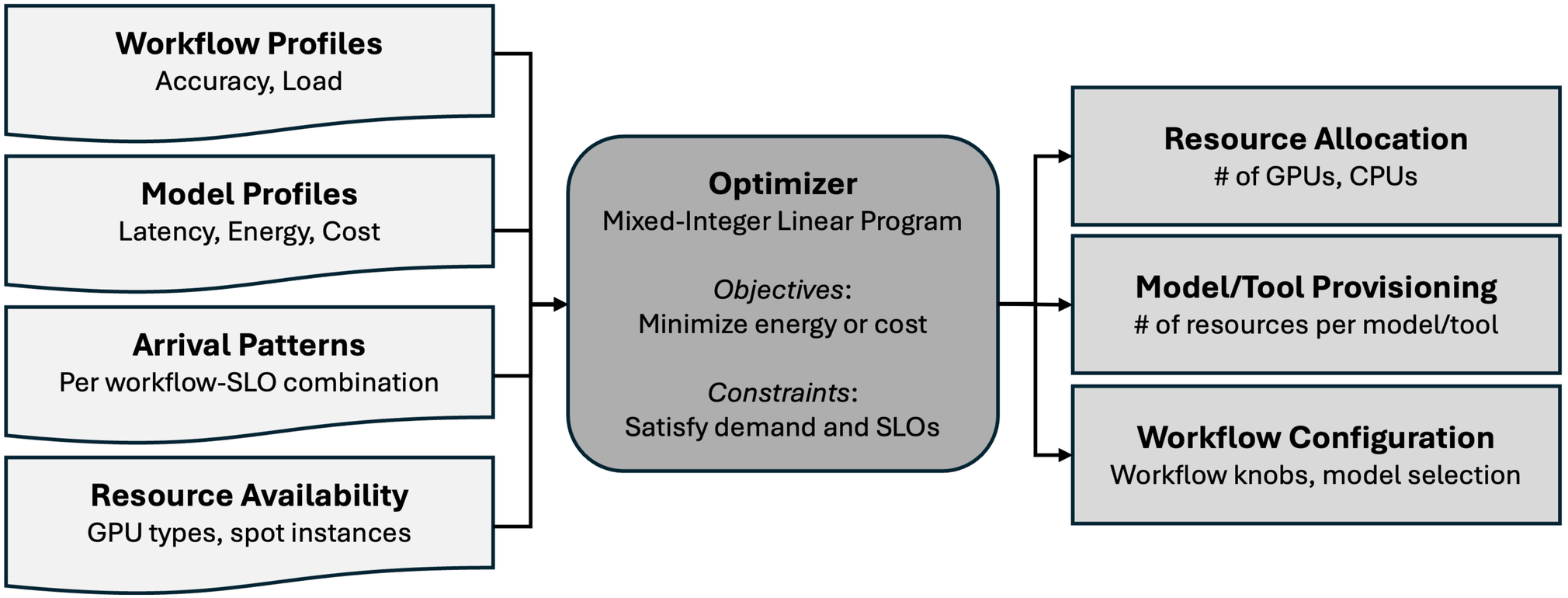
Phase 2: Optimization + Execution

Configuration is a runtime decision, not a development decision



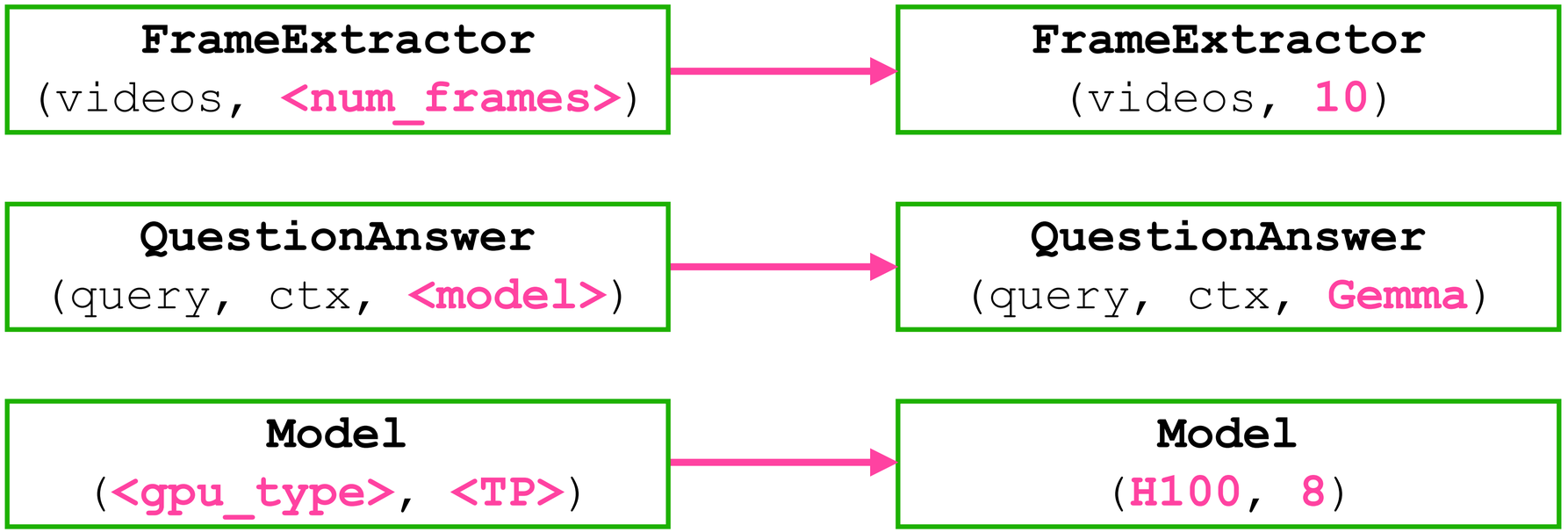
Phase 2: Optimization + Execution

Configuration is a runtime decision, not a development decision



Phase 2: Optimization + Execution

Plugging in the execution details



Phase 2: Optimization + Execution

Multi-workflow joint optimization

Phase 2: Optimization + Execution

Multi-workflow joint optimization

- Each **(workflow, SLO)-combination** has many feasible **configurations**

Phase 2: Optimization + Execution

Multi-workflow joint optimization

- Each **(workflow, SLO)-combination** has many feasible **configurations**
- Optimizer chooses **jointly**: steers different workflows towards **overlapping configurations**:
  - Same models, tools
  - Better, efficient hardware; amortizes per-user cost

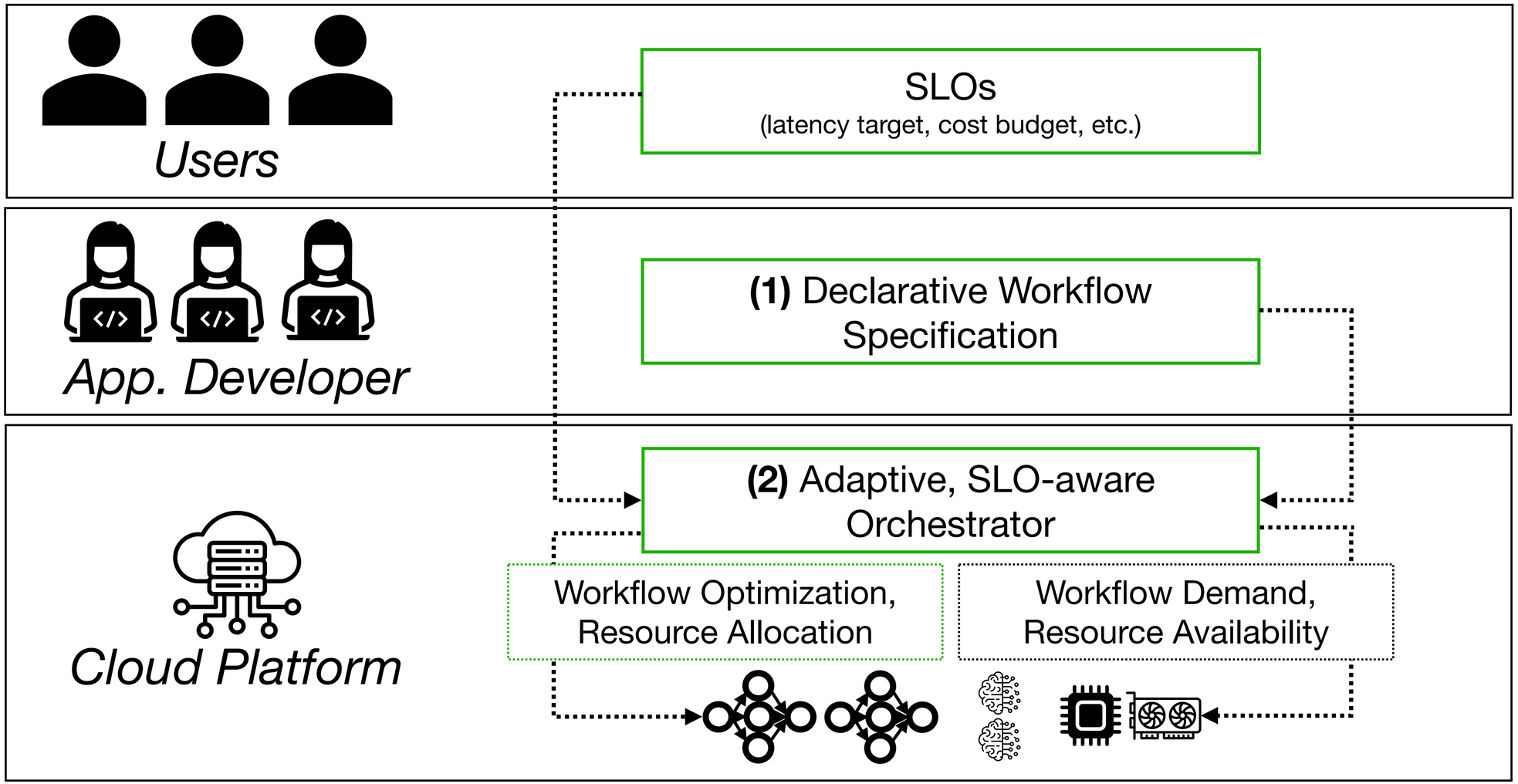
Phase 2: Optimization + Execution

Multi-workflow joint optimization

- Each **(workflow, SLO)-combination** has many feasible **configurations**
- Optimizer chooses **jointly**: steers different workflows towards **overlapping configurations**:
  - Same models, tools
  - Better, efficient hardware; amortizes per-user cost
- **Adapts** to dynamism in:
  - **Load** per (workflow, SLO)-combination
  - **Resource availability** (spot instances, energy sources etc.)

Murakkab

Cross-stack visibility + runtime orchestration across *different* workflows

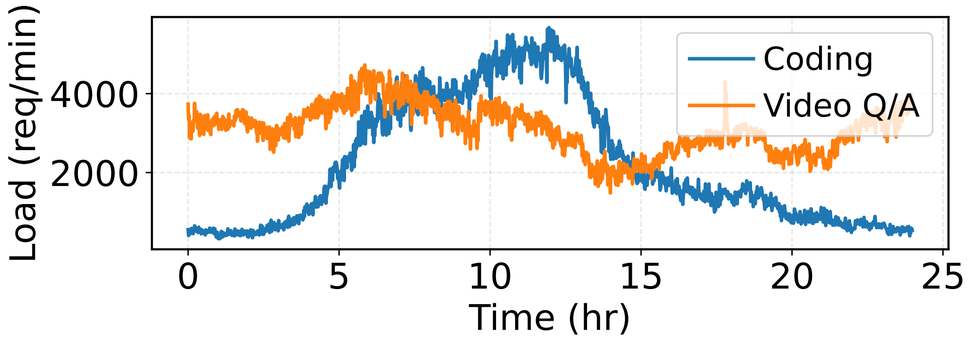


How much resource-efficiency does Murakkab unlock?

Experiment Setup

Workload:

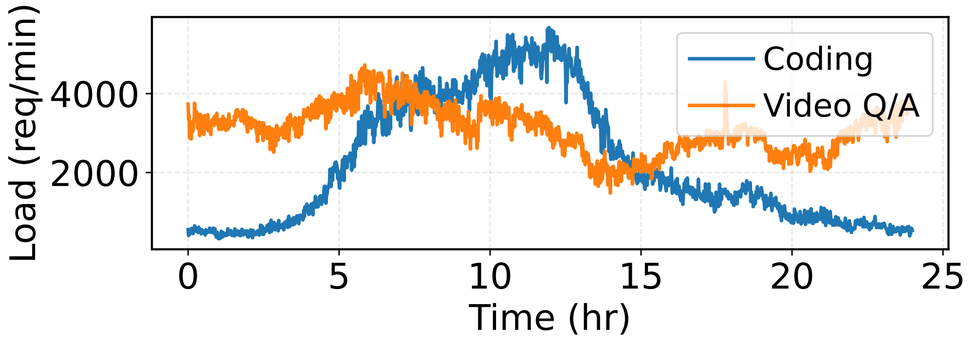
- 24 hour Azure Production LLM traces
- Video Q/A + Coding workflows
- SLO mix: 70% high-accuracy, 30% low-latency



Experiment Setup

Workload:

- 24 hour Azure Production LLM traces
- Video Q/A + Coding workflows
- SLO mix: 70% high-accuracy, 30% low-latency

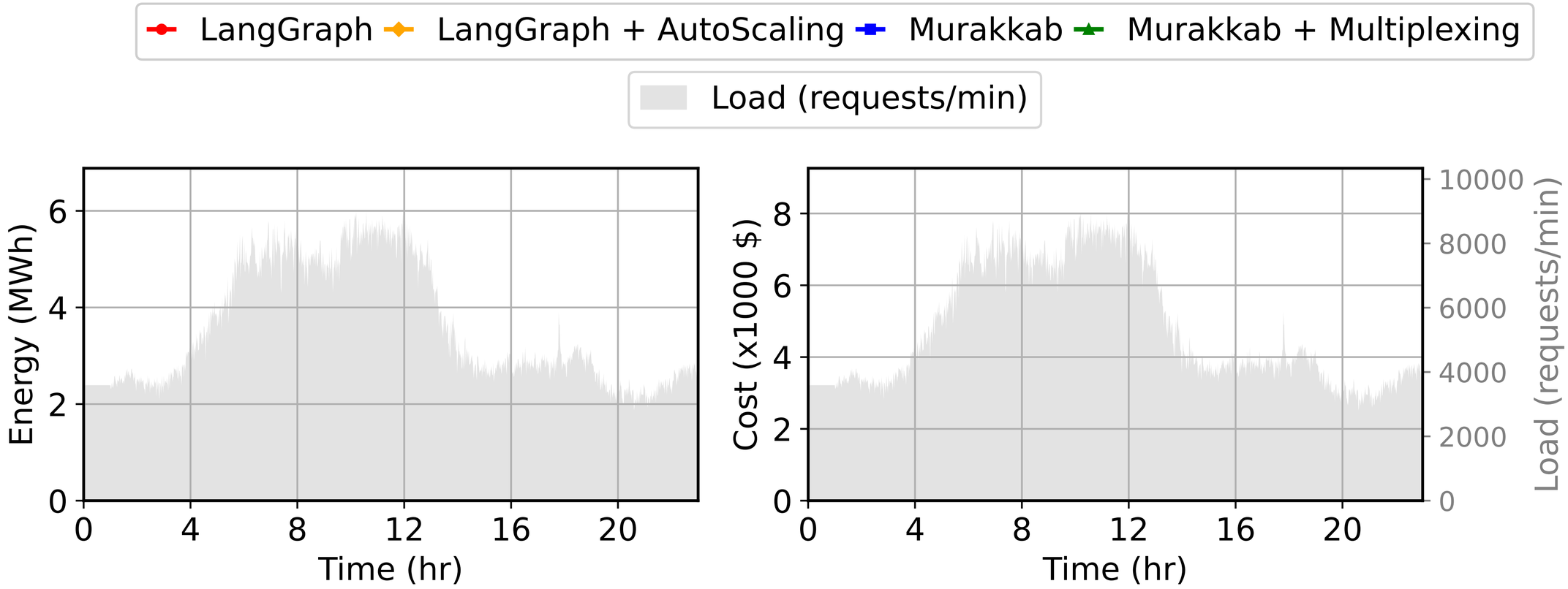


Baselines:

- LangGraph *(hand-tuned; balancing latency/accuracy; no workflow visibility, static allocation)*
- LangGraph + Auto-Scaling *(same auto-scaler as Murakkab, for fairness)*

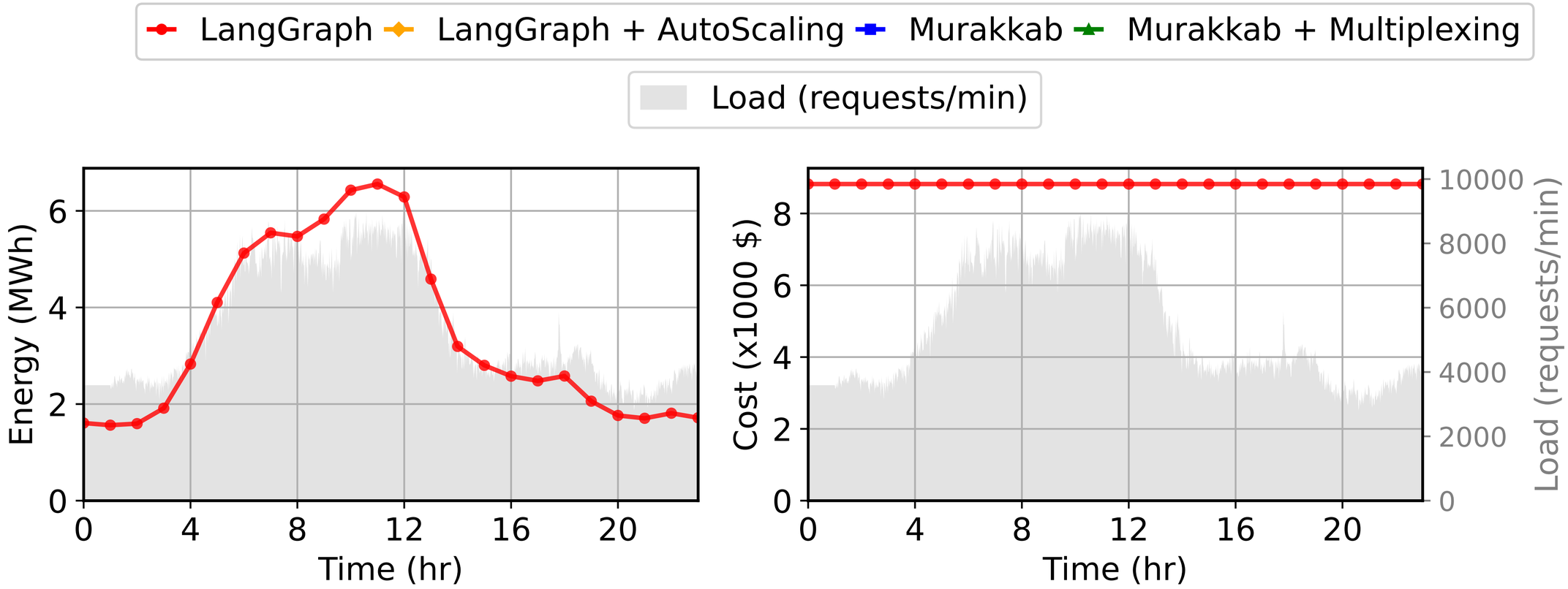
Video Q/A + Coding

SLO mix: 70% high-accuracy, 30% low-latency



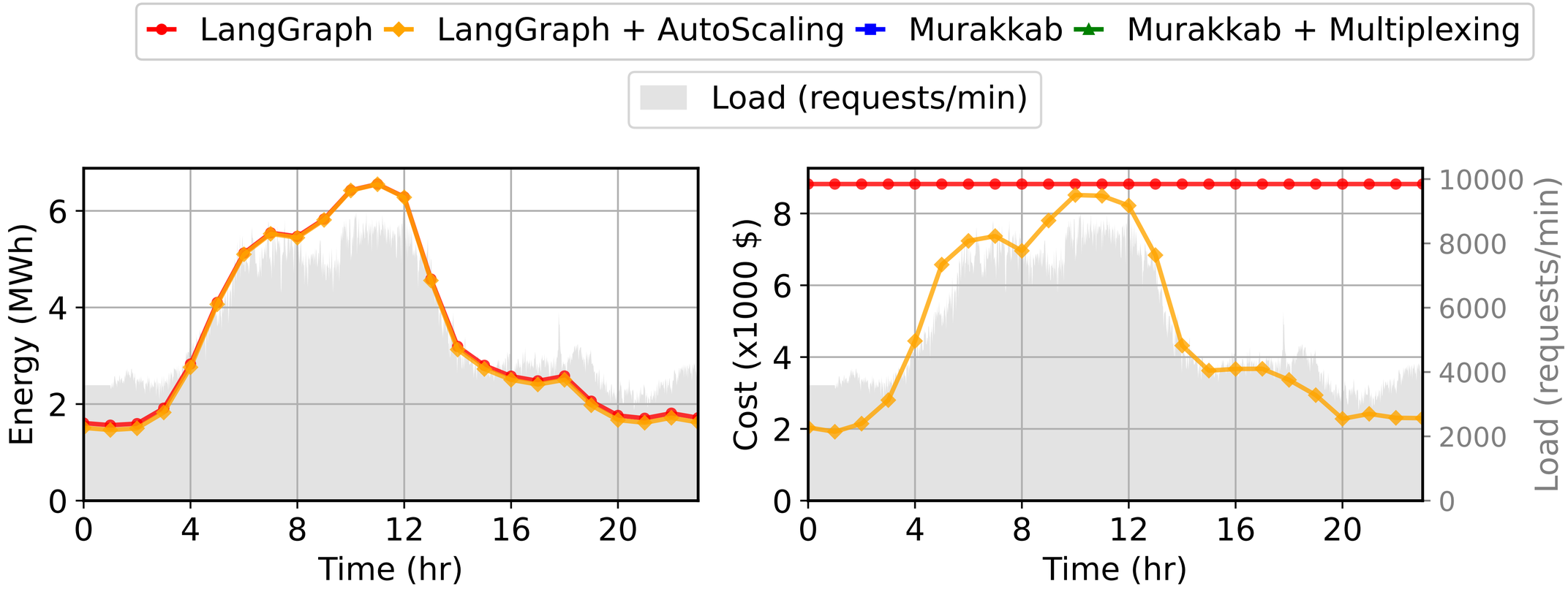
Video Q/A + Coding

SLO mix: 70% high-accuracy, 30% low-latency



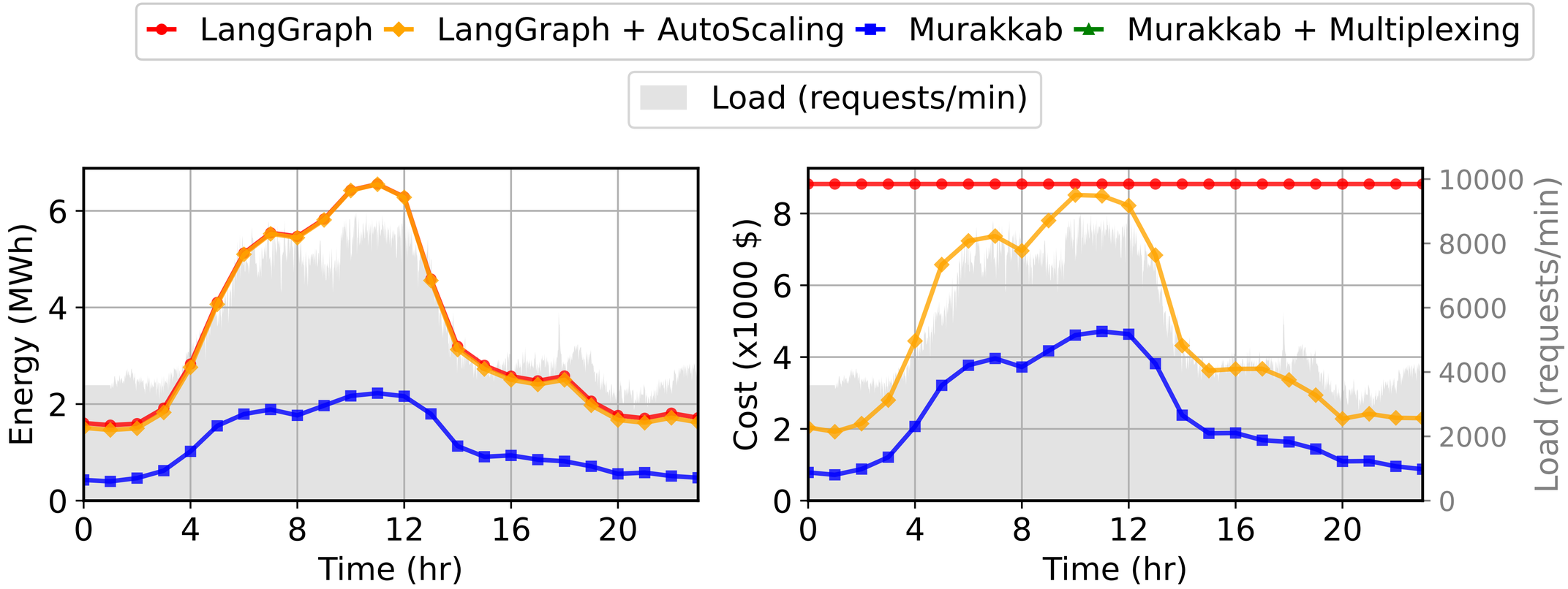
Video Q/A + Coding

SLO mix: 70% high-accuracy, 30% low-latency



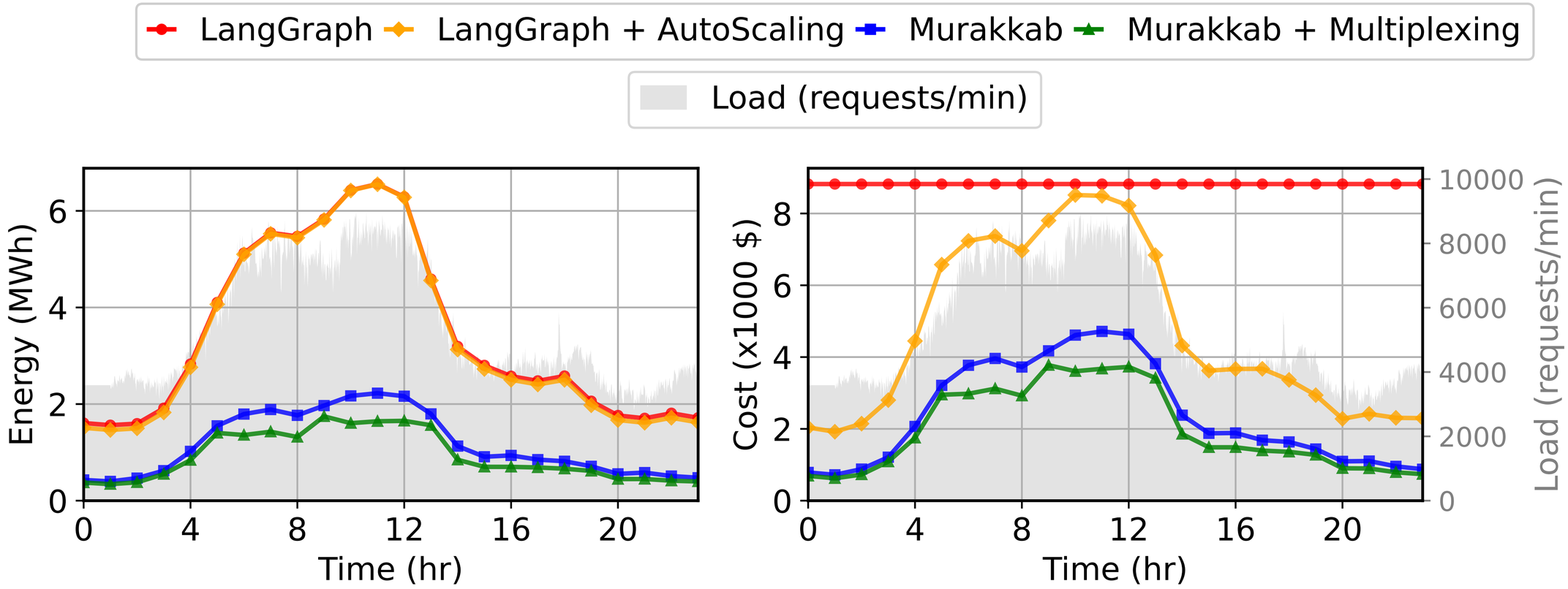
Video Q/A + Coding

SLO mix: 70% high-accuracy, 30% low-latency



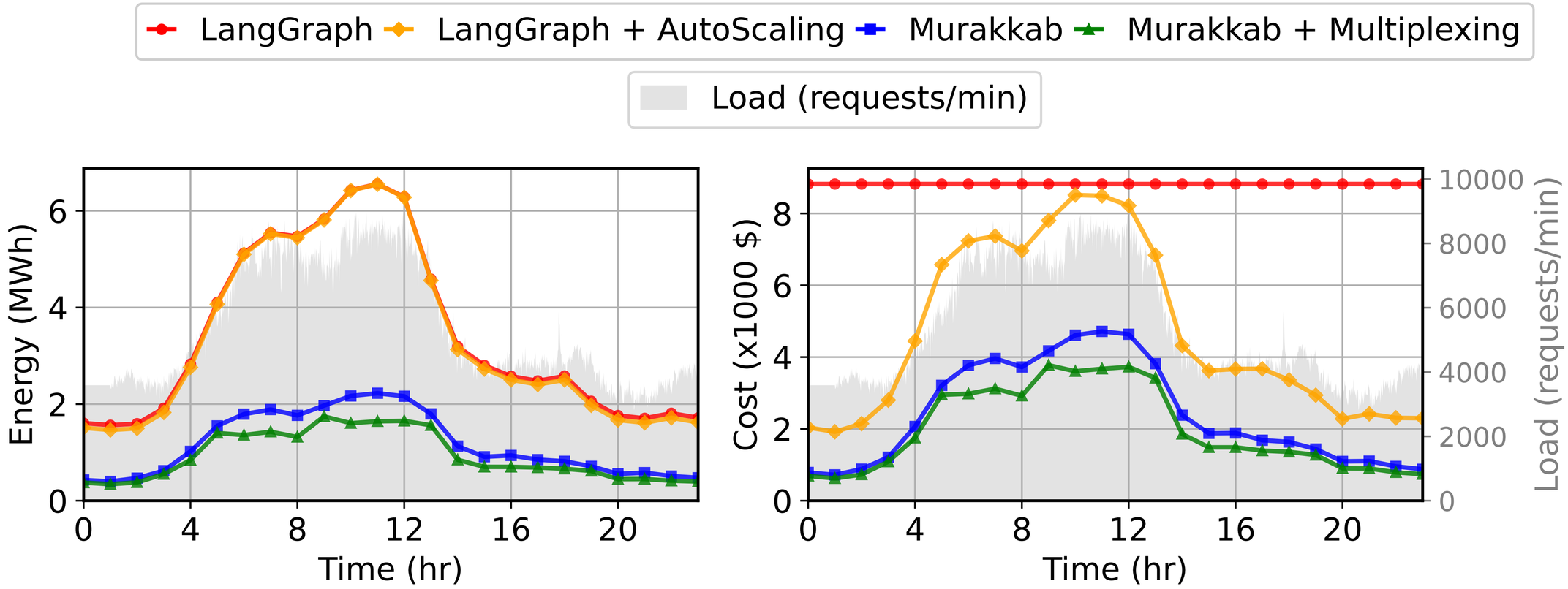
Video Q/A + Coding

SLO mix: 70% high-accuracy, 30% low-latency



Murakkab unlocks higher resource-efficiency

Up to 3.7× less energy, 4.3× lower cost, 2.8× fewer GPUs



Murakkab

Summary

- Today’s agentic workflow execution is **siloed** and loses out on the opportunity for unified, end-to-end optimization across the stack.

Murakkab

Summary

- Today's agentic workflow execution is **siloed** and loses out on the opportunity for unified, end-to-end optimization across the stack.
- **Decoupling** workflow logic from execution details opens up a vast space of automated **runtime optimizations** while satisfying SLOs.

Murakkab

Summary

- Today's agentic workflow execution is **siloed** and loses out on the opportunity for unified, end-to-end optimization across the stack.
- **Decoupling** workflow logic from execution details opens up a vast space of automated **runtime optimizations** while satisfying SLOs.
- Visibility into all layers of the stack allows **joint optimization** across different **multi-tenant workflows** in cloud platforms for greater **resource-efficiency**.