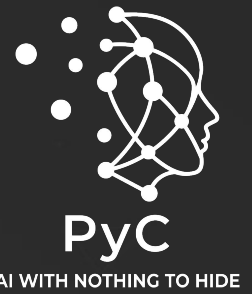
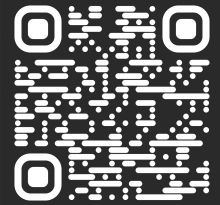
Foundations of Concept-Based Interpretable Deep Learning



Giuseppe Marra & Pietro Barbiero
giuseppe.marra@kuleuven.be pietro.barbiero@ibm.com

In collaboration with: Giovanni de Felice and Mateo Espinosa Zarlenga

ESSAI26



KU LEUVEN IBM Research

fwo Swiss National Science Foundation HASLERSTIFTUNG

Course Outline

I. Interpretability theory

II. Blueprint for interpretable models

+ interpretability in PyTorch

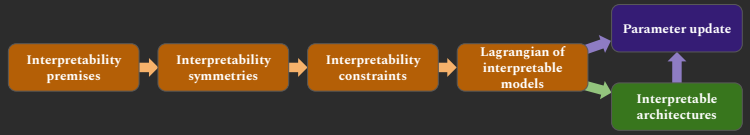
III. Concept representations

IV. Concept-based predictors

Mon 2:00 pm – 3:30 pm

Tue 2:00 pm – 3:30 pm

The Standard Interpretable Model



	Shared concept semantics	Prediction-concept dependency	Bounded reasoning
Premises			
Symmetries	$\chi(\mathfrak{g}_w \circ c_w^{[h]}) = \chi(c_w^{[h]})$	$\exists \mathfrak{g}_c \in \mathfrak{G}_c$ such that $\mathfrak{g}_c.(\nabla_z f)^\top = (\nabla_z f)^\top$	$K^{[h]}(\mathfrak{g}(\phi(c)), \ldots, \nabla^{(\nu)}\mathfrak{g}(\phi(c))) = \mu(c, \phi(c))K^{[h]}(\phi, \ldots, \nabla_c^{(\nu)}\phi) = 0$
Constraints	$\mathbb{I}_{\Delta c_w^{[h]}>0} \cdot \gamma(-\Delta c_w) = 0$	$1 - \frac{\|Q_c^\top Q_f\|_F^2}{\text{rank}(\nabla_z f)} = 0$	$K^{[h]}(\phi, \ldots, \nabla_c^{(n)}\phi) = 0$
Lagrangian	$L(\theta_{f,c,\phi}, \mathcal{D}, y, K^{[h]}) = T - V = \underbrace{\sum_{i\in\{f,c,\phi\}} \frac{1}{2} m(\partial_t \theta_i)^T(\partial_t \theta_i)}_{\text{parameter dynamics}} - \underbrace{\mathcal{L}(f(z;\theta_f), y)}_{\text{prediction error}} - \underbrace{\lambda_1 \sum_{w=1}^{m} \sum_{z_i \in \mathcal{D}} \mathbb{I}_{\Delta c_w^{[h]}(z,z_i)>0} \cdot \gamma(-\Delta c_w(z, z_i; \theta_c))}_{\text{Constraint I}} - \underbrace{\lambda_2 \left(1 - \frac{\|Q_c^\top Q_f\|_F^2}{\text{rank}(\nabla_z f)}\right)}_{\text{Constraint II}} - \underbrace{K^{[h]}(\phi(c(z;\theta_c);\theta_\phi), \ldots, \nabla_c^{(n)}\phi(c(z;\theta_c);\theta_\phi))}_{\text{Constraint III}}$		
Equations of motion	$m\frac{\partial^2 \theta_f}{\partial t^2} = -\nabla_{\theta_f}\mathcal{L}(f(z;\theta_f), y) - \lambda_2 \left(1 - \frac{\|Q_c^\top Q_f\|_F^2}{\text{rank}(\nabla_z f)}\right)$	$m\frac{\partial^2 \theta_c}{\partial t^2} = -\lambda_1 \sum_{\substack{w=1,\ldots,m \\ z_i \in \mathcal{D}}} \mathbb{I}_{\Delta c_w^{[h]}>0} \nabla_{\theta_c} \gamma(-\Delta c_w(z, z_i; \theta_c)) - \lambda_2 \nabla_{\theta_c}\left(1 - \frac{\|Q_c^\top Q_f\|_F^2}{\text{rank}(\nabla_z f)}\right) - \lambda_3 \nabla_{\theta_c} K^{[h]}(\phi(c;\theta_\phi), \ldots, \nabla_c^{(n)}\phi(c;\theta_\phi))$	$m\frac{\partial^2 \theta_\phi}{\partial t^2} = -\lambda_3 \nabla_{\theta_\phi} K^{[h]}(\phi(c;\theta_\phi), \ldots, \nabla_c^{(n)}\phi(c;\theta_\phi))$
Architectures	$c_w(z) = \sum_{i=1}^{N} \beta_i(z) \left(\sum_{k=1}^{N} \theta_k \mathbb{I}_{c_w^{[h]}(z_i) \geq c_w^{[h]}(z_k)}\right)$	$f = \phi(c_1, \ldots, c_l)$	$\phi_\theta(c) = \sum_{k=1}^{n} \theta_k \psi_k(c), \quad \phi_\theta(c) \in \ker(K^{[h]})$
General solution		$\underbrace{f = \phi_\theta^{[h]}}_{\text{Constraint II}} \left(\left[\underbrace{\sum_{i=1}^{N} \beta_i(z) \left(\sum_{k=1}^{N} \theta_k \mathbb{I}_{c_w^{[h]}(z_i) \geq c_w^{[h]}(z_k)}\right)}_{\text{Constraint I}} \right]_{w=1}^{l} \right), \quad \underbrace{\phi_\theta^{[h]} \in \ker(K^{[h]})}_{\text{Constraint III}}$	

The Standard Interpretable Model

Premises	**Shared concept semantics**		**Bounded reasoning**
Symmetries	$\chi(\mathfrak{g}_w \circ c_w^{[h]}) = \chi(c_w^{[h]})$		$K^{[h]}(\mathfrak{g}(\phi(c)), \ldots, \nabla_c^{(\nu)}\mathfrak{g}(\phi(c))) = \mu(c,\phi(c))K^{[h]}(\phi,\ldots,\nabla_c^{(\nu)}\phi) = 0$
Constraints	$\mathbb{I}_{\Delta c_w^{[h]}>0} \cdot \gamma(-\Delta c_w) = 0$		$K^{[h]}(\phi, \ldots, \nabla_c^{(n)}\phi) = 0$
Lagrangian	$L(\theta_{f,c,\phi}, \mathcal{D}, y, K^{[h]}) = T - V = \underbrace{\sum_{i\in\{f,c,\phi\}} \frac{1}{2} m(\partial_t \theta_i)^2}_{\text{parameter dynami}}$		$\underbrace{-\lambda_2\left(1 - \frac{\|Q_c^\top Q_f\|_F^2}{\text{rank}(\nabla_z f)}\right)}_{\text{Constraint II}} \underbrace{- K^{[h]}(\phi(c(z;\theta_c);\theta_\phi), \ldots, \nabla_c^{(n)}\phi(c(z;\theta_c);\theta_\phi))}_{\text{Constraint III}}$
Equations of motion	$m\frac{\partial^2\theta_f}{\partial t^2} = -\nabla_{\theta_f}\mathcal{L}(f(z;\theta_f), y) - \lambda_2\left(1 - \frac{\|Q_c^\top Q_f\|_F^2}{\text{rank}(\nabla_z f)}\right)$		$m\frac{\partial^2\theta_\phi}{\partial t^2} = -\lambda_3 \nabla_{\theta_\phi} K^{[h]}(\phi(c;\theta_\phi), \ldots, \nabla_c^{(n)}\phi(c;\theta_\phi))$
Architectures	$c_w(z) = \sum_{i=1}^{N} \beta_i(z)\left(\sum_{k=1}^{N} \theta_k \mathbb{I}_{c_w^{[h]}(z_i)\geq c_w^{[h]}(z_k)}\right)$	$f = \phi(c_1, \ldots, c_l)$	$\phi_\theta(c) = \sum_{k=1}^{n} \theta_k \psi_k(c), \quad \phi_\theta(c) \in \ker(K^{[h]})$
General solution		$\underbrace{f = \phi_\theta^{[h]}}_{\text{Constraint II}} \left(\left[\underbrace{\sum_{i=1}^{N} \beta_i(z)\left(\sum_{k=1}^{N} \theta_k \mathbb{I}_{c_w^{[h]}(z_i)\geq c_w^{[h]}(z_k)}\right)}_{\text{Constraint I}}\right]_{w=1}^{l}\right), \quad \underbrace{\phi_\theta^{[h]} \in \ker(K^{[h]})}_{\text{Constraint III}}$	

PyC

AI WITH NOTHING TO HIDE

From Theory to Coding

I. Annotated Tensors (~5 mins)

II. Blueprint for interpretable models (~10 mins)

III. Interventions & control (~15 mins)

IV. PyC hands-on! Interpretable layers and interventions (~10 mins)

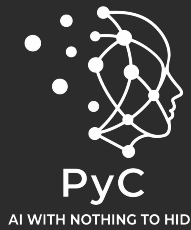
Q&A + 5 min break

V. Blueprint for probabilistic interpretable machine learning (~15 mins)

VI. PyC hands-on! Probabilistic interpretable models (~10 mins)

VII. High-level APIs and Conceptarium (~10 mins)

Final Q&A



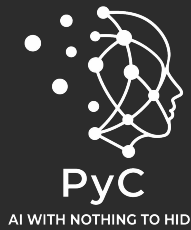
Out-of-the-box models
(High-level API)

Research in interpretable
architectures
(Mid-level API)

Research in interpretable
modules
(Low-level API)

External
libraries

PyTorch



Out-of-the-box models
(High-level API)

Research in interpretable
architectures
(Mid-level API)

Research in interpretable
modules
(Low-level API)

Annotations

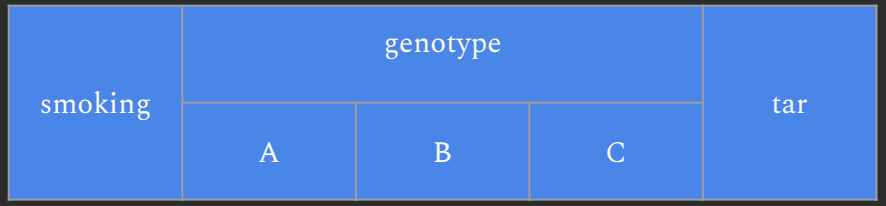
Annotated
tensors

External
libraries

PyTorch

Annotated tensors

Annotations: define concept names and domains




```
annotations = pyc.Annotations(
    labels=["smoking", "genotype", "tar"],
    cardinalities=[1, 3, 1],
    types=["binary", "categorical", "continuous"],
)
```

Annotated tensors

Annotations: define concept names and domains
Annotated tensors: enable concept-based operations on torch tensors

smoking	genotype			tar
	A	B	C	
0	0	1	0	-3.1
1	1	0	0	4.2
0	1	0	0	2.3

```python
annotations = pyc.Annotations(
    labels=["smoking", "genotype", "tar"],
    cardinalities=[1, 3, 1],
    types=["binary", "categorical", "continuous"],
)

tensor = pyc.AnnotatedTensor(
    data=torch.randn(10, 5),  # (batch_size, sum(cardinalities))
    annotation=annotations,
)

smoking = tensor["smoking"]  # slice by concept name
binary = tensor.split_by_type("binary")  # slice by concept type
```

From Theory to Coding

I. Annotated Tensors (~5 mins)

II. Blueprint for interpretable models (~10 mins)

III. Interventions & control (~15 mins)

IV. PyC hands-on! Interpretable layers and interventions (~10 mins)

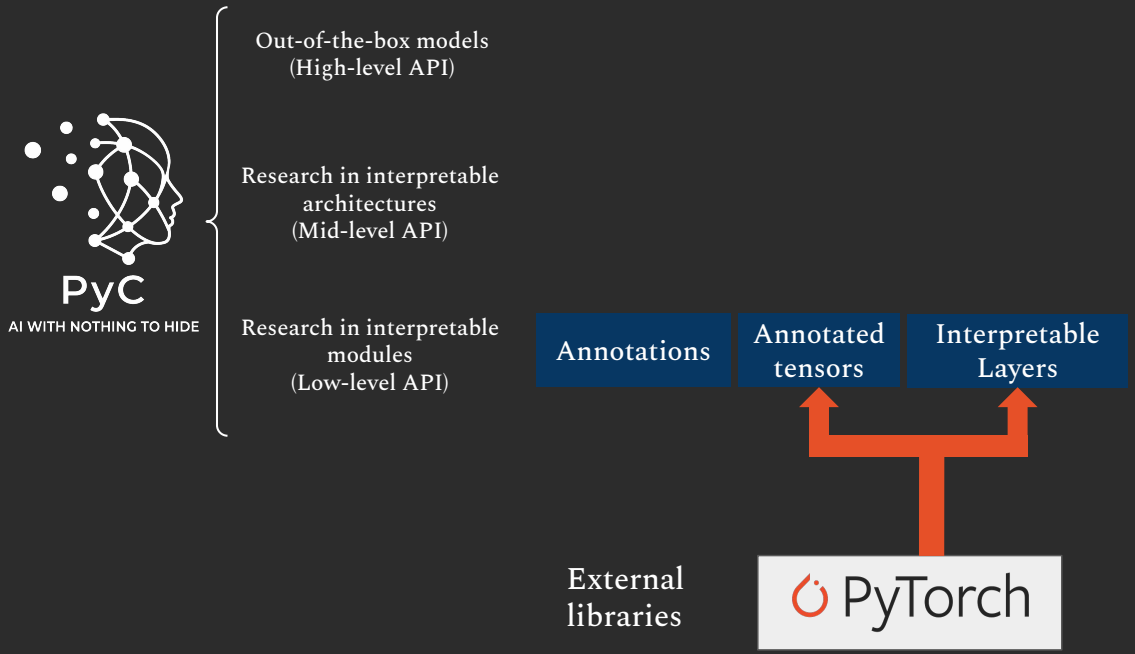
Q&A + 5 min break

V. Blueprint for probabilistic interpretable machine learning (~15 mins)

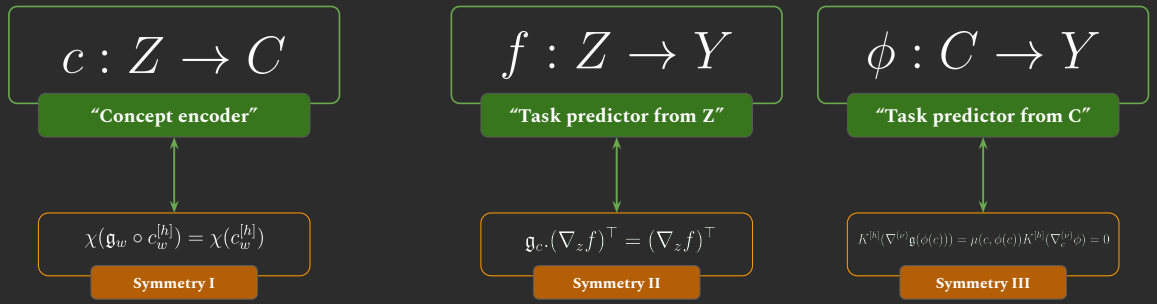
VI. PyC hands-on! Probabilistic interpretable models (~10 mins)

VII. High-level APIs and Conceptarium (~10 mins)

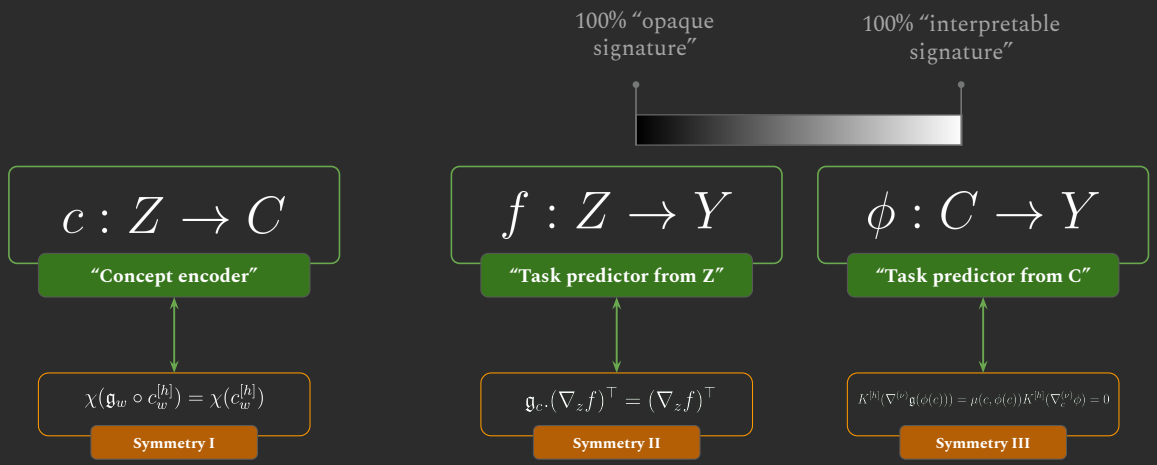
Final Q&A


PyC
AI WITH NOTHING TO HIDE
Out-of-the-box models
(High-level API)
Research in interpretable
architectures
(Mid-level API)
Research in interpretable
modules
(Low-level API)
Annotations
Annotated
tensors
Interpretable
Layers
External
libraries
PyTorch

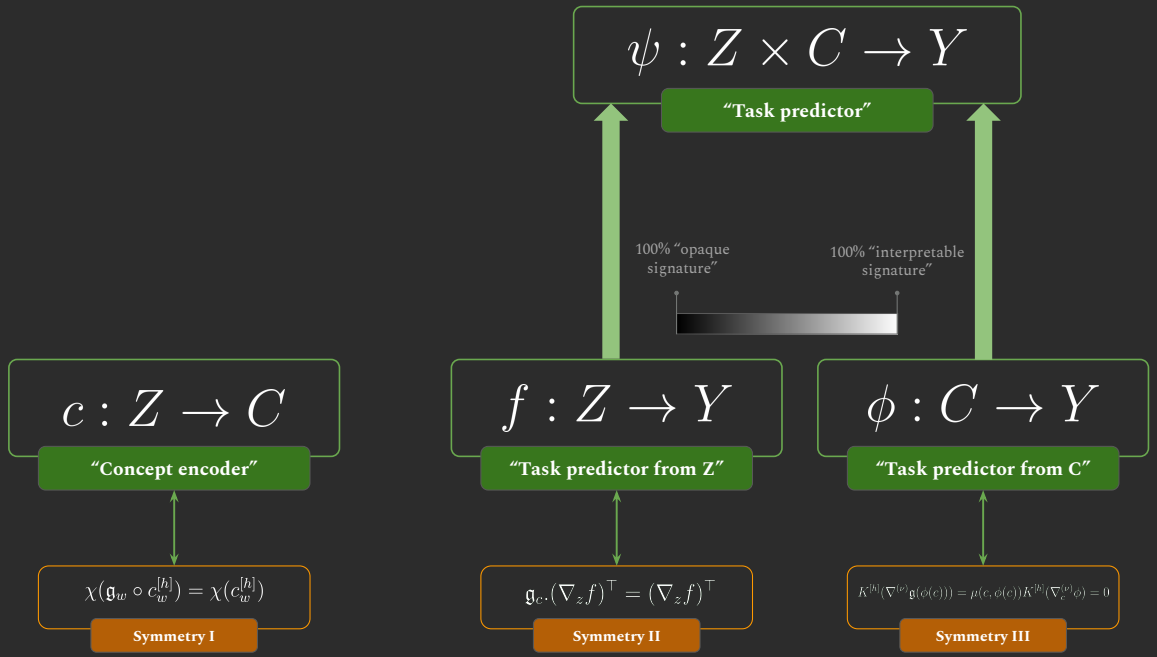
Building blocks of interpretable models



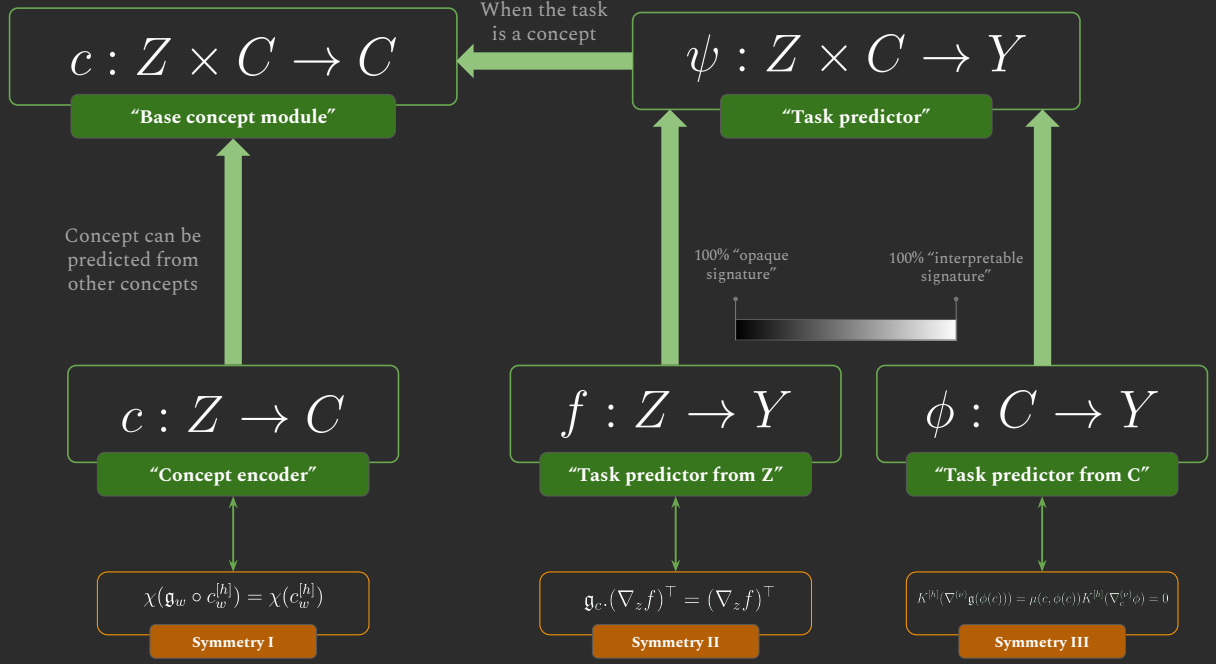
Building blocks of interpretable models



Building blocks of interpretable models

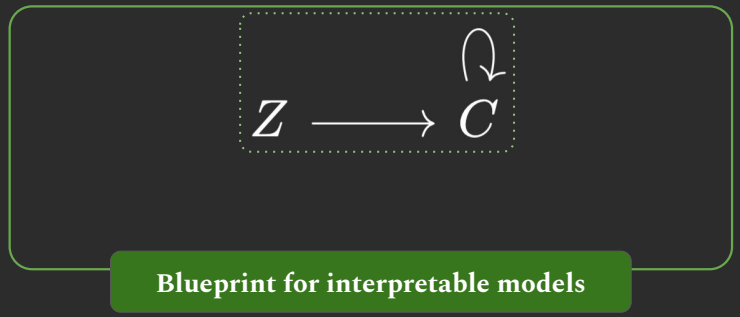


Building blocks of interpretable models

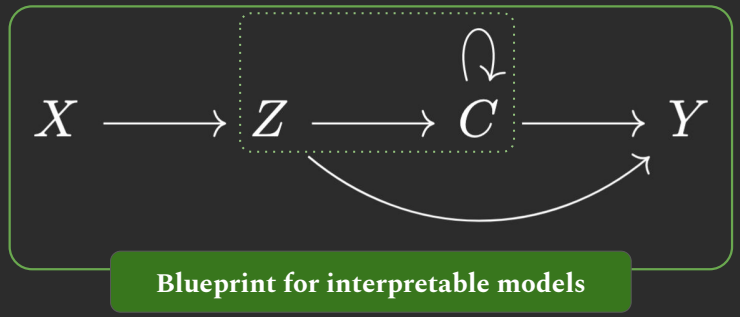



```python
class BaseConceptLayer(ABC, torch.nn.Module):
    """Abstract base class for concept layers...."""
    def __init__(
        self,
        out_concepts: Union[int, AxisAnnotation],
        in_concepts: Union[int, AxisAnnotation] = None,
        in_embeddings: Union[int, AxisAnnotation] = None,
        *args,
        **kwargs,
    ):
```

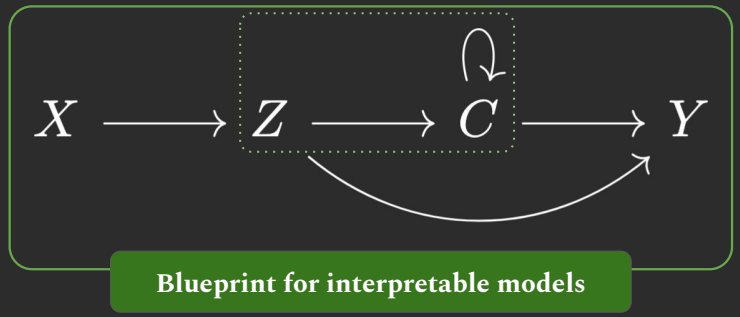
Blueprint for interpretable models



Blueprint for interpretable models

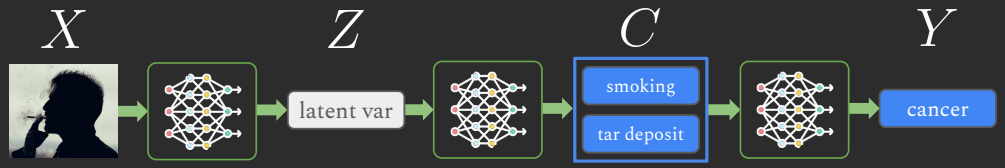


Blueprint for interpretable models




```python
embedding_encoder = torch.nn.Sequential(
    torch.nn.Linear(n_features, embedding_dims),
    torch.nn.LeakyReLU(),
)
c_encoder = pyc.nn.LinearEmbeddingToConcept(
    in_embeddings=embedding_dims,
    out_concepts=annotations
)
y_predictor = pyc.nn.LinearConceptToConcept(
    in_concepts=concept_dims,
    out_concepts=task_dims
)
model = torch.nn.ModuleDict({
    "embedding_encoder": embedding_encoder,
    "concept_encoder": c_encoder,
    "task_predictor": y_predictor
})
embeddings = model["embedding_encoder"](x_train)
c_pred = model["concept_encoder"](embeddings)
y_pred = model["task_predictor"](c_pred)
```

CONCEPT BOTTLENECK MODEL



[1] Koh, et al. "Concept bottleneck models." International conference on machine learning. PMLR, 2020.

From Theory to Coding

I. Annotated Tensors (~5 mins)

II. Blueprint for interpretable models (~10 mins)

III. Interventions & control (~15 mins)

IV. PyC hands-on! Interpretable layers and interventions (~10 mins)

Q&A + 5 min break

V. Blueprint for probabilistic interpretable machine learning (~15 mins)

VI. PyC hands-on! Probabilistic interpretable models (~10 mins)

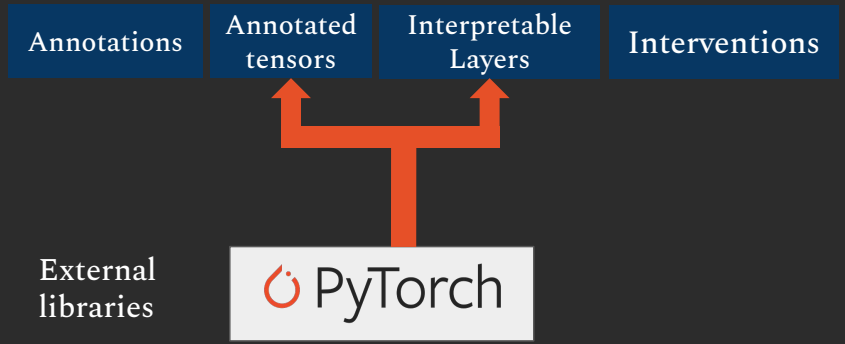
VII. High-level APIs and Conceptarium (~10 mins)

Final Q&A

Out-of-the-box models
(High-level API)

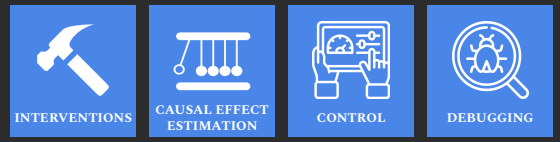
Research in interpretable
architectures
(Mid-level API)

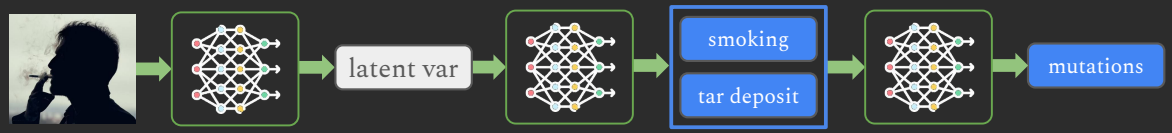
Research in interpretable
modules
(Low-level API)



Interventions & control

INTERPRETABLE MODELS SUPPORT

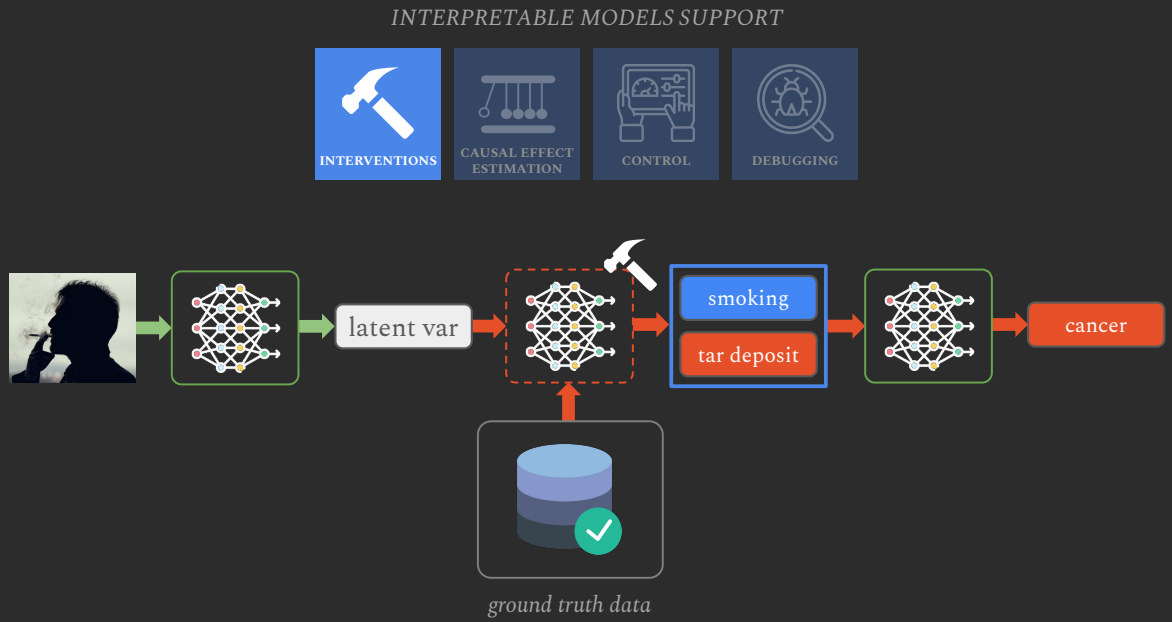





```python
embedding_encoder = torch.nn.Sequential(
    torch.nn.Linear(n_features, embedding_dims),
    torch.nn.LeakyReLU(),
)
c_encoder = pyc.nn.LinearEmbeddingToConcept(
    in_embeddings=embedding_dims,
    out_concepts=annotations
)
y_predictor = pyc.nn.LinearConceptToConcept(
    in_concepts=concept_dims,
    out_concepts=task_dims
)
model = torch.nn.ModuleDict({
    "embedding_encoder": embedding_encoder,
    "concept_encoder": c_encoder,
    "task_predictor": y_predictor
})
embeddings = model["embedding_encoder"](x_train)
c_pred = model["concept_encoder"](embeddings)
y_pred = model["task_predictor"](c_pred)
```

[1] Koh, et al. "Concept bottleneck models." International conference on machine learning. PMLR, 2020.

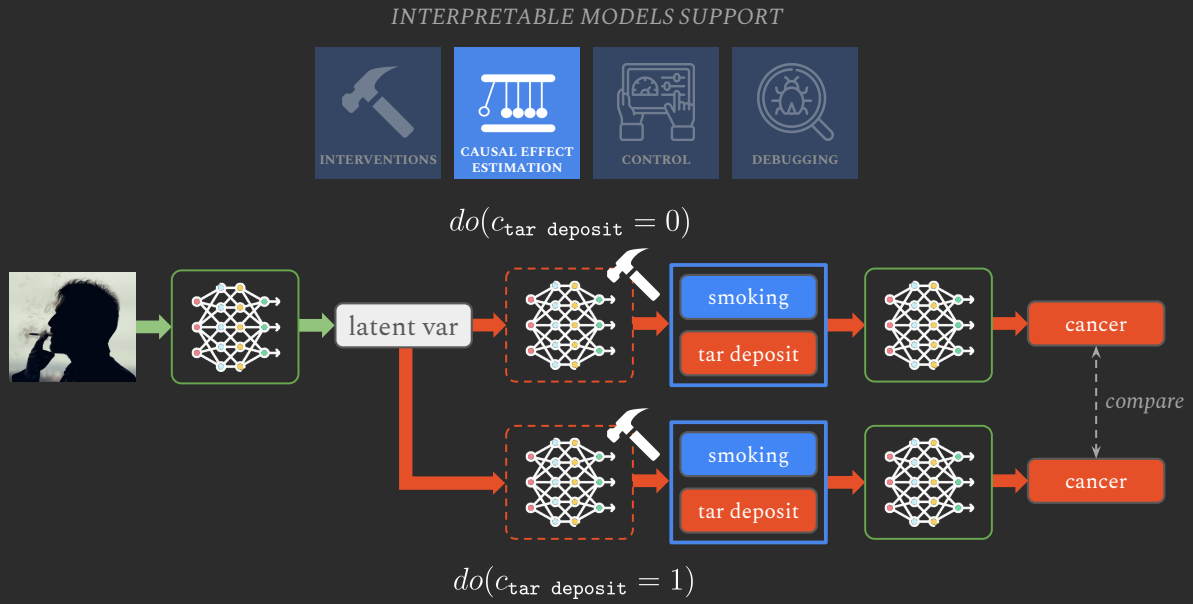
Interventions & control




```
concept_encoder_intervened = pyc.nn.InterventionModule(
    model["concept_encoder"],
    intervention_strategy=pyc.nn.GroundTruthIntervention(c_train),
    intervention_policy=pyc.nn.UniformPolicy(),
    out_concepts_to_intervene_on=["tar"]
)
c_pred_intervened = concept_encoder_intervened(embeddings)
```

[1] Koh, et al. "Concept bottleneck models." International conference on machine learning. PMLR, 2020.

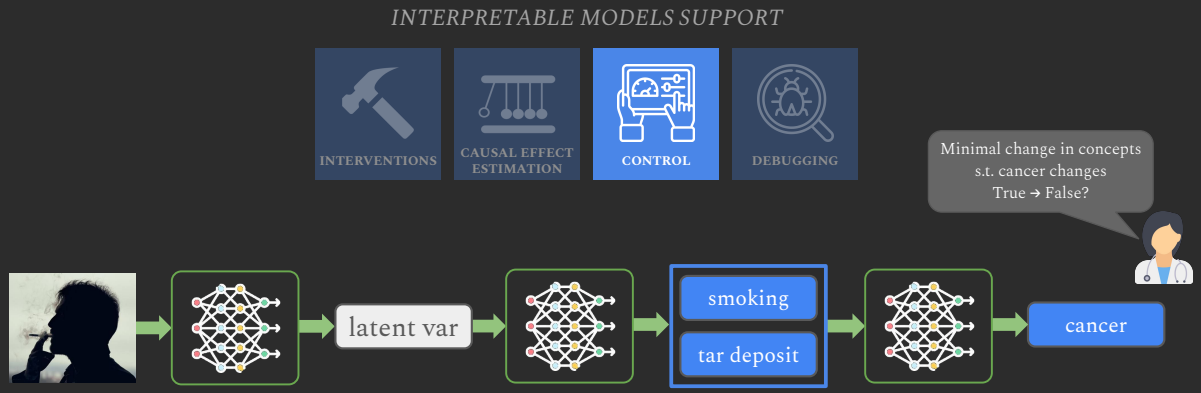
Interventions & control



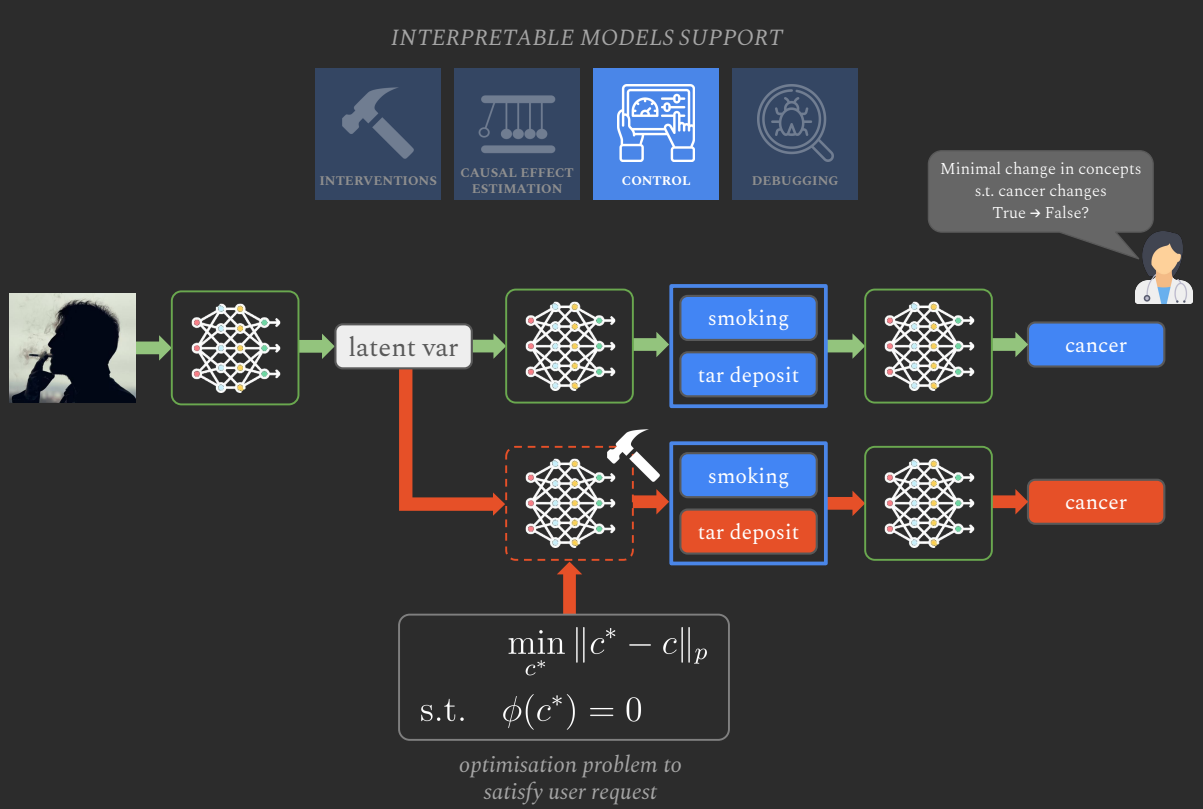

```python
concept_encoder_intervened_1 = pyc.nn.InterventionModule(
    model["concept_encoder"],
    intervention_strategy=pyc.nn.DoIntervention(constants=1.0),
    intervention_policy=pyc.nn.UniformPolicy(),
    out_concepts_to_intervene_on=["tar"]
)
c_pred_intervened_1 = concept_encoder_intervened_1(embeddings)

concept_encoder_intervened_0 = pyc.nn.InterventionModule(
    model["concept_encoder"],
    intervention_strategy=pyc.nn.DoIntervention(constants=0.0),
    intervention_policy=pyc.nn.UniformPolicy(),
    out_concepts_to_intervene_on=["tar"]
)
c_pred_intervened_0 = concept_encoder_intervened_0(embeddings)

avg_treatment_effect = pyc.nn.functional.cace_score(
    c_pred_intervened_0,
    c_pred_intervened_1
)
```

[1] Goyal et al. "Explaining classifiers with causal concept effect (cace)." arXiv preprint arXiv:1907.07165 (2019).

Interventions & control



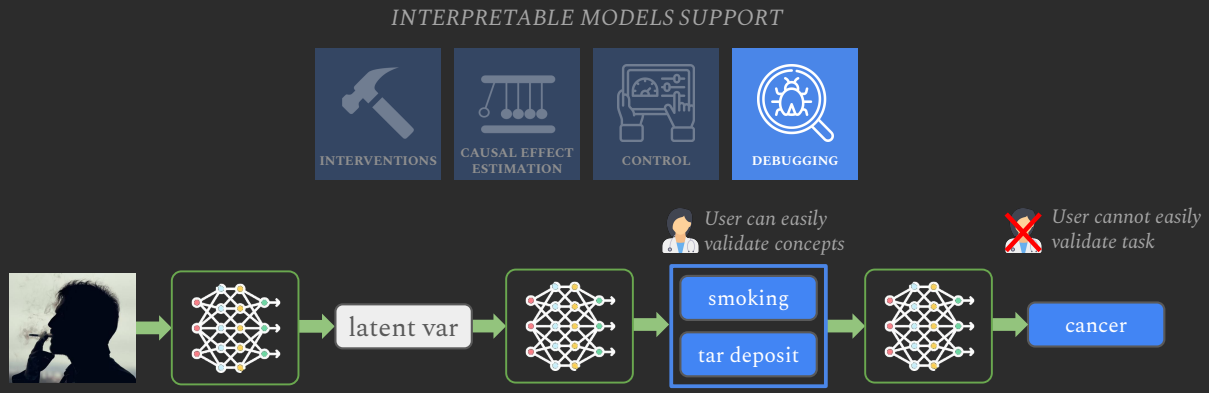
Interventions & control



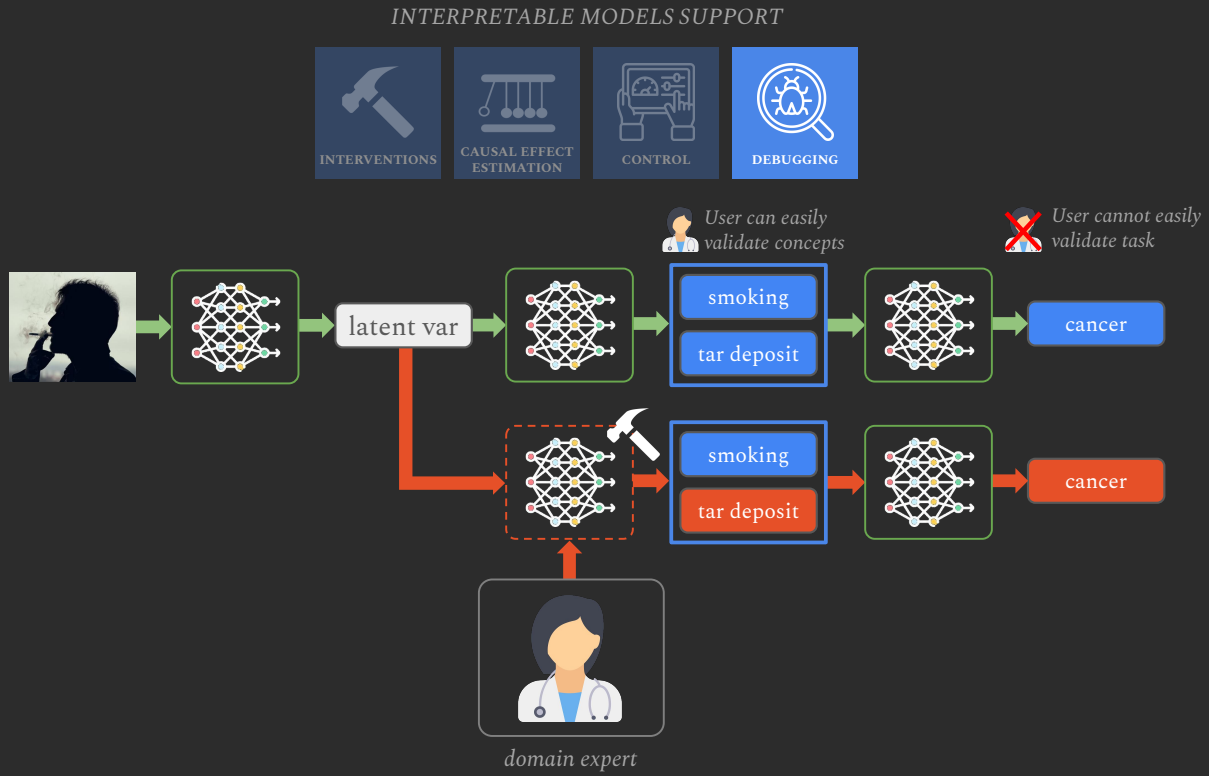

```python
y_target = torch.ones(batch_size, out_module_size)
concept_encoder_intervened = pyc.nn.InterventionModule(
    model["concept_encoder"],
    pyc.nn.ContrastiveIntervention(
        out_target_tensor=y_target,
        x_to_out_module=model["task_predictor"],
    ),
    pyc.nn.UniformPolicy(),
)
c_pred_contrastive = concept_encoder_intervened(
    embeddings[:batch_size]
)
y_pred_contrastive = model["task_predictor"](
    c_pred_contrastive
)
```

[1] Wachter et al. "Counterfactual explanations without opening the black box: Automated decisions and the GDPR." arXiv preprint arXiv:1711.00399 (2017).

Interventions & control



Interventions & control

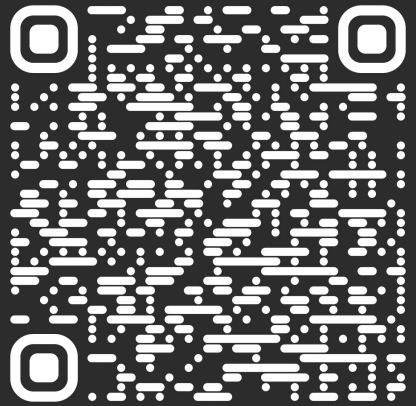



```
concept_encoder_intervened = pyc.nn.InterventionModule(
    model["concept_encoder"],
    intervention_strategy=pyc.nn.GroundTruthIntervention(c_train),
    intervention_policy=pyc.nn.UniformPolicy(),
    out_concepts_to_intervene_on=["tar"]
)
c_pred_intervened = concept_encoder_intervened(embeddings)
```

[1] Koh, et al. "Concept bottleneck models." International conference on machine learning. PMLR, 2020.

Hands-on session

Create a copy of the notebook



https://colab.research.google.com/drive/1oB5UEItyQu4EaAr-IJIWX8eCzJs3uAg9?usp=sharing

From Theory to Coding

I. Annotated Tensors (~5 mins)

II. Blueprint for interpretable models (~10 mins)

III. Interventions & control (~15 mins)

IV. PyC hands-on! Interpretable layers and interventions (~10 mins)

Q&A + 5 min break

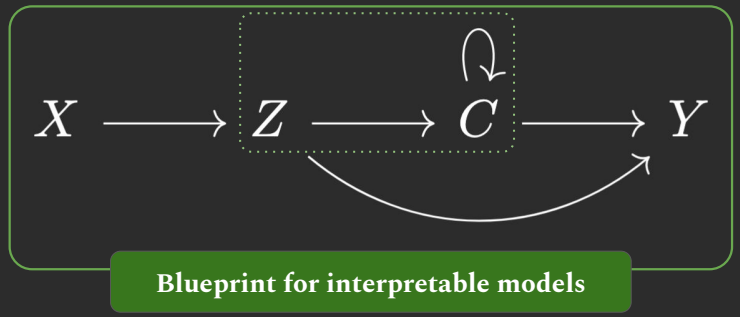
V. Blueprint for probabilistic interpretable machine learning (~15 mins)

VI. PyC hands-on! Probabilistic interpretable models (~10 mins)

VII. High-level APIs and Conceptarium (~10 mins)

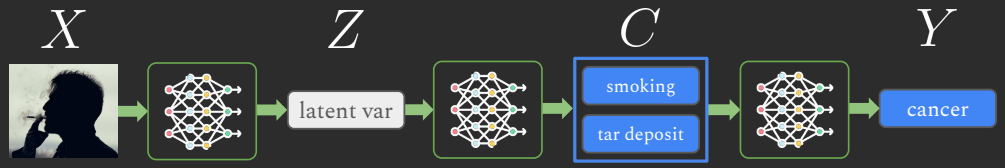
Final Q&A

Blueprint for interpretable models

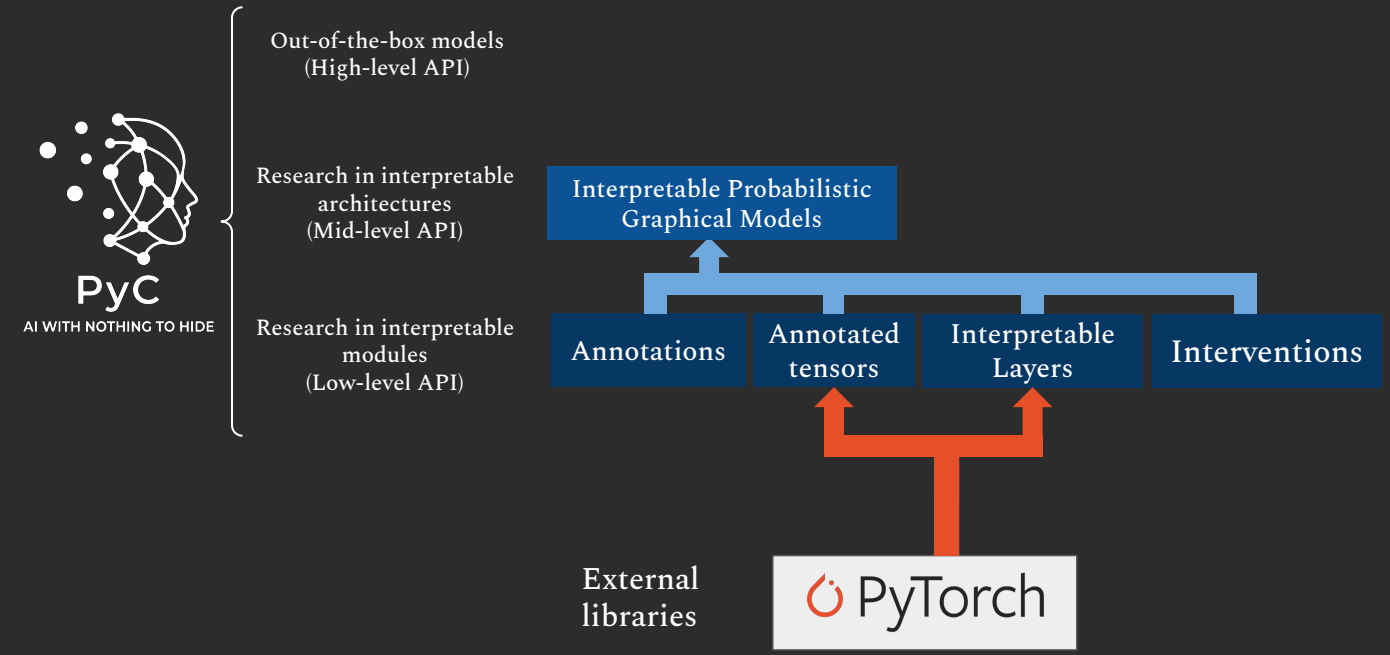



```python
embedding_encoder = torch.nn.Sequential(
    torch.nn.Linear(n_features, embedding_dims),
    torch.nn.LeakyReLU(),
)
c_encoder = pyc.nn.LinearEmbeddingToConcept(
    in_embeddings=embedding_dims,
    out_concepts=annotations
)
y_predictor = pyc.nn.LinearConceptToConcept(
    in_concepts=concept_dims,
    out_concepts=task_dims
)
model = torch.nn.ModuleDict({
    "embedding_encoder": embedding_encoder,
    "concept_encoder": c_encoder,
    "task_predictor": y_predictor
})
embeddings = model["embedding_encoder"](x_train)
c_pred = model["concept_encoder"](embeddings)
y_pred = model["task_predictor"](c_pred)
```

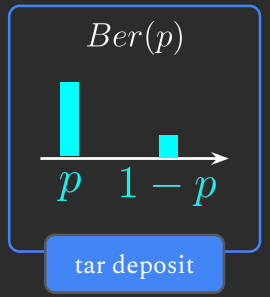
CONCEPT BOTTLENECK MODEL



[1] Koh, et al. "Concept bottleneck models." International conference on machine learning. PMLR, 2020.

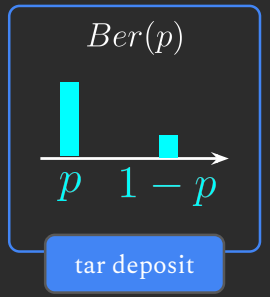

PyC
AI WITH NOTHING TO HIDE
Out-of-the-box models
(High-level API)
Research in interpretable
architectures
(Mid-level API)
Research in interpretable
modules
(Low-level API)
Interpretable Probabilistic
Graphical Models
Annotations
Annotated
tensors
Interpretable
Layers
Interventions
External
libraries
PyTorch

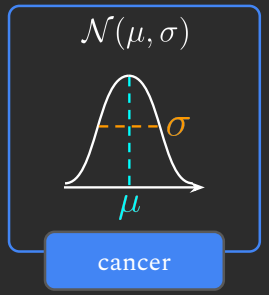
Random variables




```
tar = pyc.ConceptVariable(
    names: "tar",
    distribution=Bernoulli
)
```

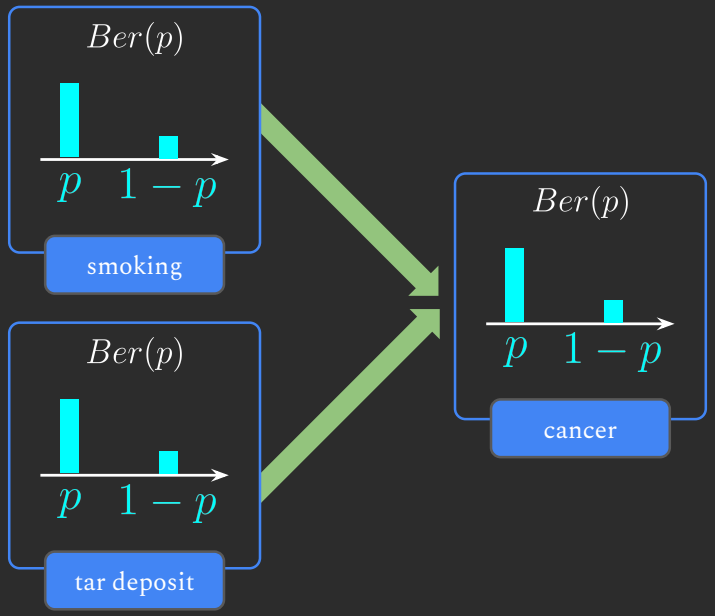
Random variables





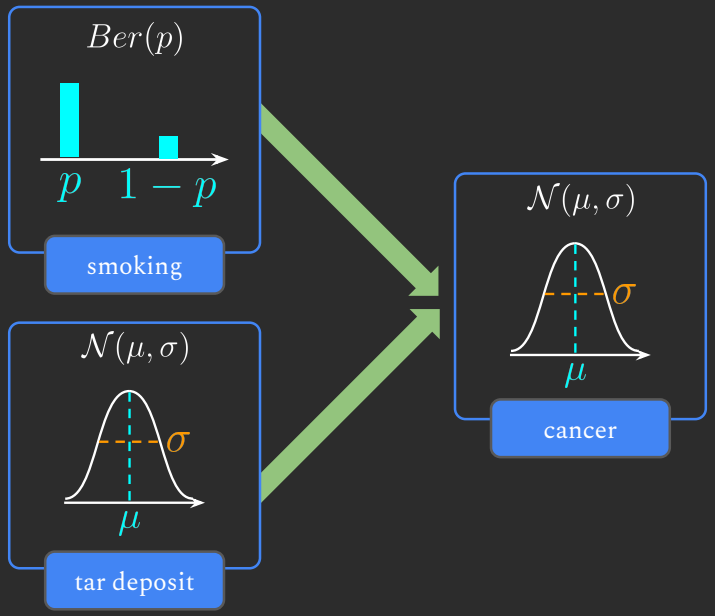

```
tar = pyc.ConceptVariable(
    names: "tar",
    distribution=Bernoulli
)
mutations = pyc.ConceptVariable(
    names: "cancer",
    distribution=Normal
)
```

Conditional probability tables



smoking	tar deposit	$\mathbb{P}(y_{\texttt{cancer}} = 1 \mid c_{\texttt{smoking}}, c_{\texttt{tar deposit}})$
0	0	0.1
0	1	0.6
1	0	0.2
1	1	0.8

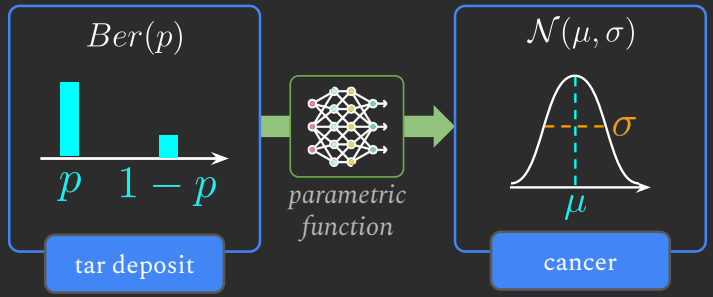
Conditional probability tables



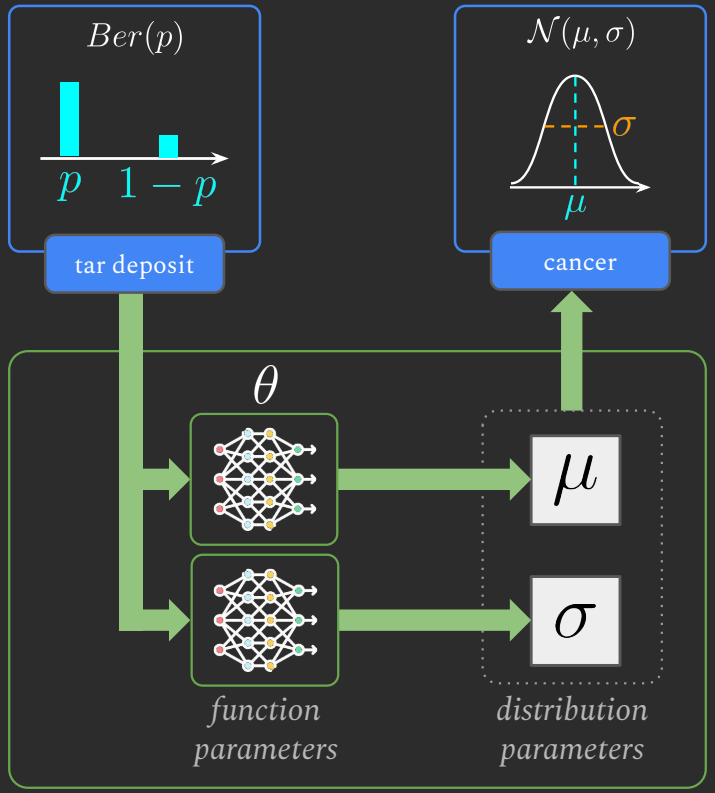
smoking	posit	$\mathbb{P}(y_{\text{ca}}$	1 $\mid c_{\texttt{smoking}}, c_{\texttt{tar deposit}})$
0	0		0.1
0	1		0.6
1	0		0.2
1	1		0.8

Solution: compute the parameters of the output distribution as a function of the parents' values!

Parametric conditional probability distributions

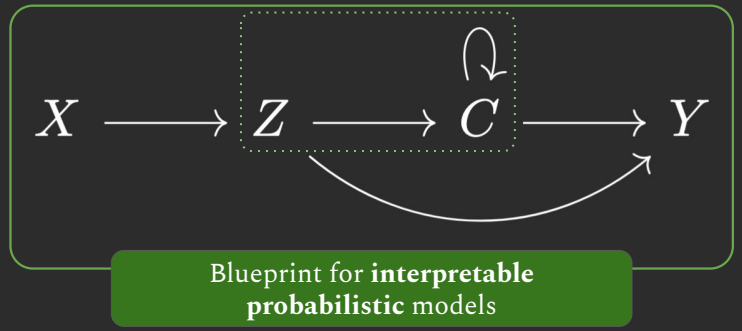


Parametric conditional probability distributions

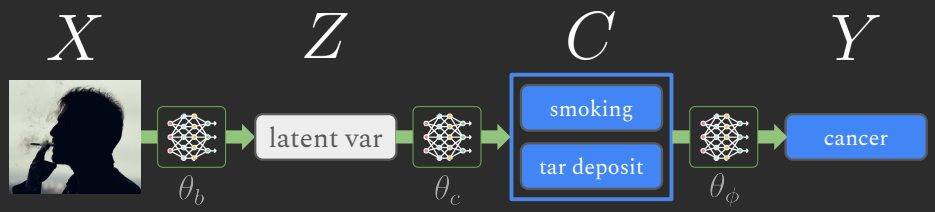



```python
cancer_cpd = pyc.nn.ParametricCPD(
    cancer,
    parents=[tar, smoking],
    parametrization={
        "loc": pyc.nn.LinearConceptToConcept(in_concepts=2, out_concepts=1),
        "scale": pyc.nn.Sequential(
            *args: pyc.nn.LinearConceptToConcept(in_concepts=2, out_concepts=1),
            torch.nn.Softplus()
        )
    }
)
```

Interpretable probabilistic models

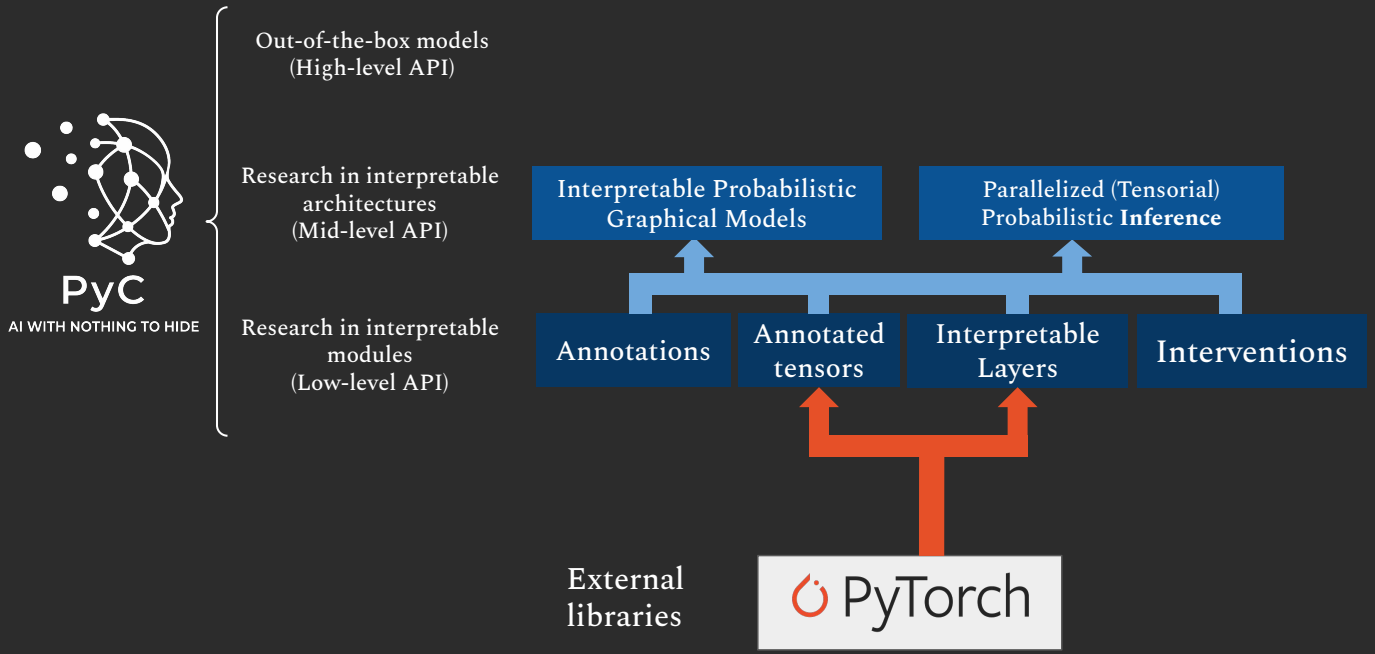


PROBABILISTIC CONCEPT BOTTLENECK MODEL

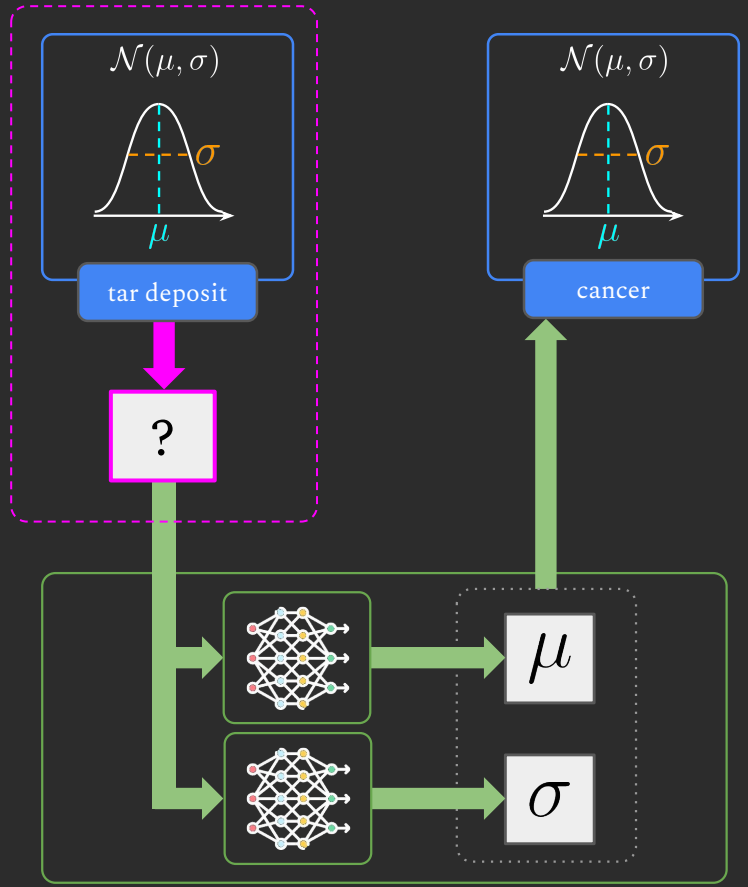


$$p(y, c \mid x; \theta_\phi, \theta_c, \theta_b) = \int p(y \mid c; \theta_\phi) p(c \mid z; \theta_c) p(z \mid x; \theta_b) \mathrm{d}z$$

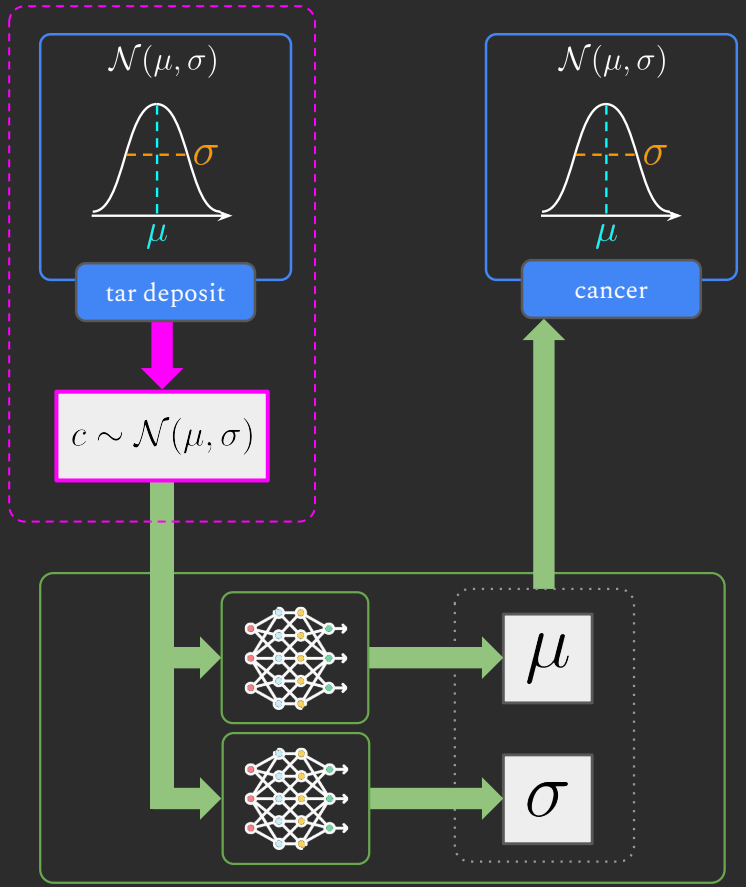
```python
x_var = pyc.nn.EmbeddingVariable(
    names: "x",
    distribution=Delta,
    size=n_features
)
c_var = pyc.nn.ConceptVariable(
    names: "concepts",
    members=["tar", "smoking"],
    distribution=Bernoulli
)
y_var = pyc.nn.ConceptVariable( names: "cancer", distribution=Normal)

input_cpd = pyc.nn.ParametricCPD(
    x_var, parents=[],
    parametrization=pyc.nn.LearnablePrior(size=1)
)
concepts_cpd = pyc.nn.ParametricCPD(
    c_var, parents=[x_var],
    parametrization={
        'logits': pyc.nn.LinearEmbeddingToConcept(n_features, out_concepts=2)
    }
)
cancer_cpd = pyc.nn.ParametricCPD(
    cancer,
    parents=[c_var],
    parametrization={
        "loc": pyc.nn.LinearConceptToConcept(in_concepts=2, out_concepts=1),
        "scale": pyc.nn.Sequential(
            *args: pyc.nn.LinearConceptToConcept(in_concepts=2, out_concepts=1),
            torch.nn.Softplus()
        )
    }
)

model = pyc.nn.BayesianNetwork(
    variables=[x_var, c_var, y_var],
    factors=[input_cpd, concepts_cpd, cancer_cpd]
)
```


PyC
AI WITH NOTHING TO HIDE
Out-of-the-box models
(High-level API)
Research in interpretable
architectures
(Mid-level API)
Research in interpretable
modules
(Low-level API)
Interpretable Probabilistic
Graphical Models
Parallelized (Tensorial)
Probabilistic **Inference**
Annotations
Annotated
tensors
Interpretable
Layers
Interventions
External
libraries
PyTorch

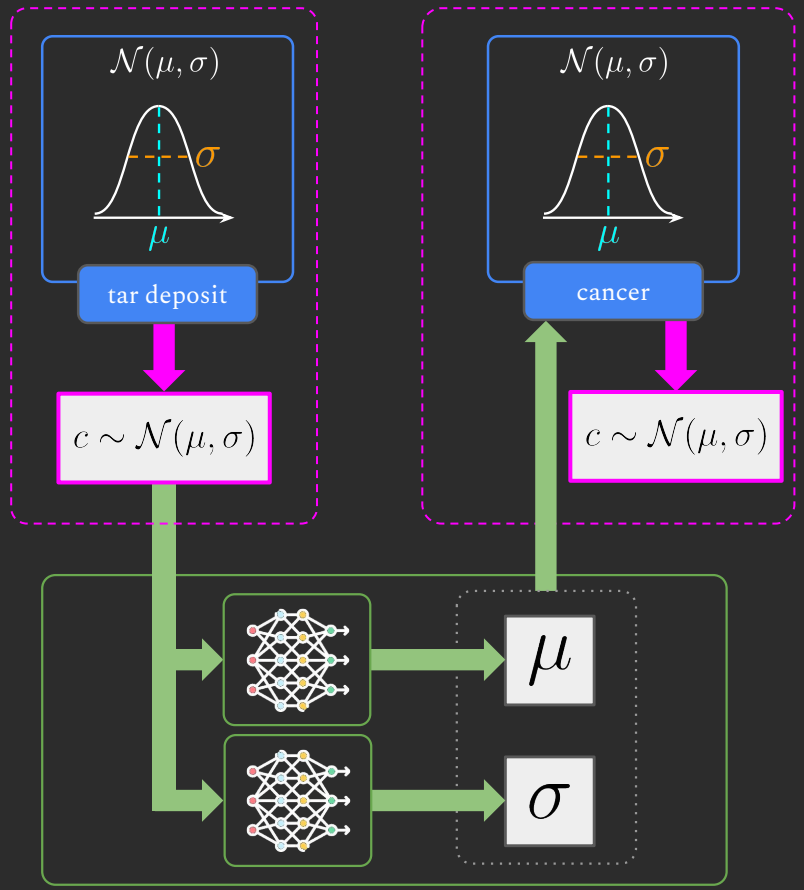
Probabilistic inference



Ancestral sampling

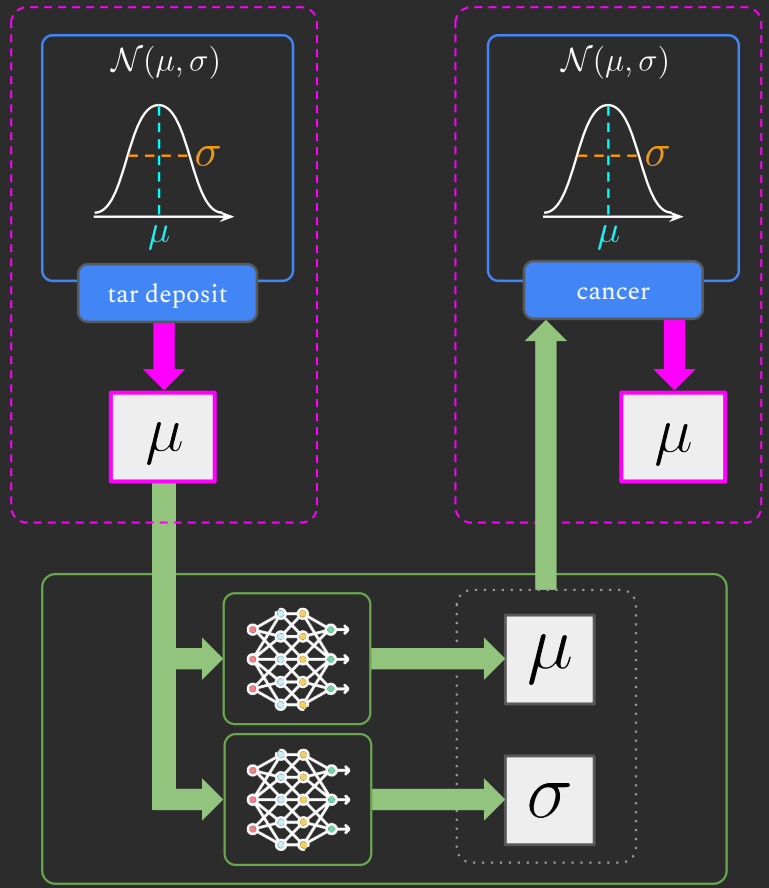


Ancestral sampling




```python
model = pyc.nn.BayesianNetwork(
    variables=[x_var, c_var, y_var],
    factors=[input_cpd, concepts_cpd, cancer_cpd]
)

inference = pyc.nn.AncestralSamplingInference(model)
cancer_pred_sampled = inference.query(
    query=["cancer"],
    evidence={
        "tar": c_true["tar"].tensor,
    }
)

inference = pyc.nn.DeterministicInference(model)
cancer_pred_deterministic = inference.query(
    query=["cancer"],
    evidence={
        "tar": c_true["tar"].tensor,
    }
)
```

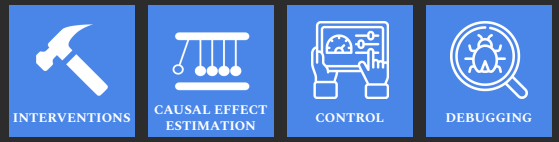
“Deterministic” inference

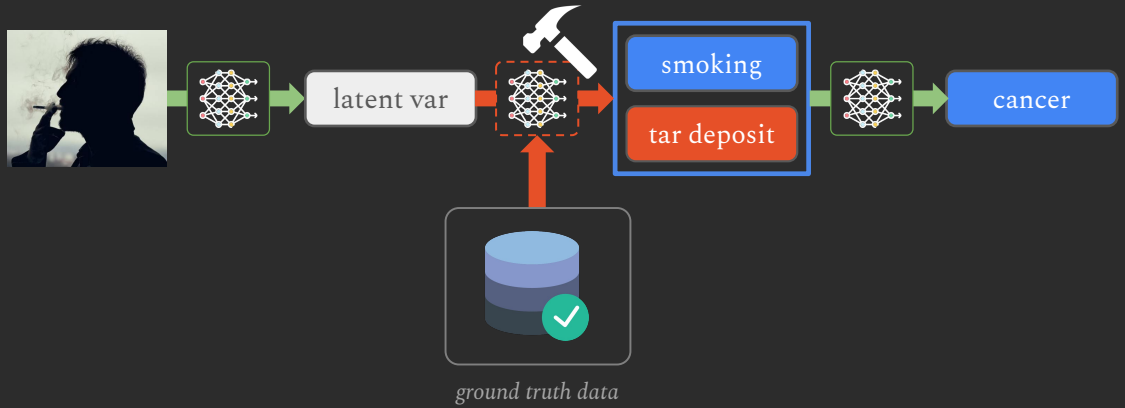



```python
model = pyc.nn.BayesianNetwork(
    variables=[x_var, c_var, y_var],
    factors=[input_cpd, concepts_cpd, cancer_cpd]
)

inference = pyc.nn.AncestralSamplingInference(model)
cancer_pred_sampled = inference.query(
    query=["cancer"],
    evidence={
        "tar": c_true["tar"].tensor,
    }
)

inference = pyc.nn.DeterministicInference(model)
cancer_pred_deterministic = inference.query(
    query=["cancer"],
    evidence={
        "tar": c_true["tar"].tensor,
    }
)
```

Interventional inference

INTERPRETABLE MODELS SUPPORT

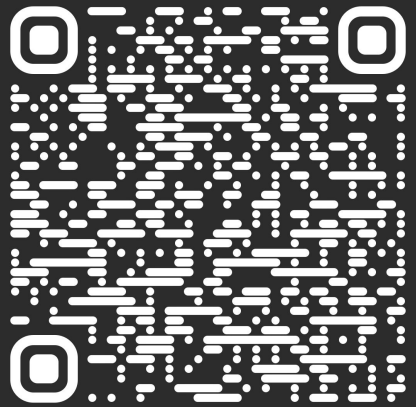





```python
with pyc.nn.intervention(
        model,
        pyc.nn.GroundTruthIntervention(c_true["tar"].tensor),
        pyc.nn.UniformPolicy(),
        variable_to_intervene_on="concepts",
        parameter_to_intervene_on="logits",
        members_to_intervene_on=["tar"]
):
    cancer_pred_intervened = inference.query(
        query=["cancer"],
        evidence={
            "tar": c_true["tar"].tensor,
        }
    )
```

Hands-on session

Create a copy of the notebook



https://colab.research.google.com/drive/1FnTUV06xz1mdQbX2qhH2o0OLm3bwl0Gv?usp=sharing

From Theory to Coding

I. Annotated Tensors (~5 mins)

II. Blueprint for interpretable models (~10 mins)

III. Interventions & control (~15 mins)

IV. PyC hands-on! Interpretable layers and interventions (~10 mins)

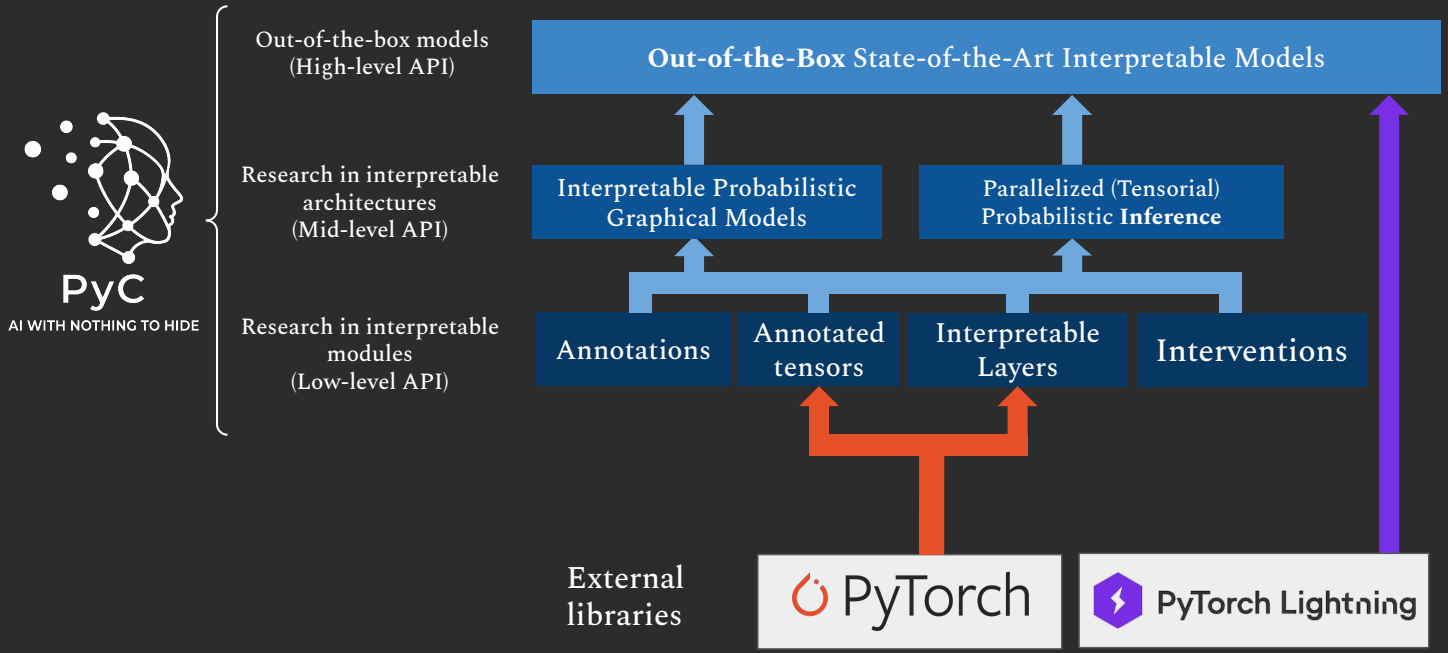
Q&A + 5 min break

V. Blueprint for probabilistic interpretable machine learning (~15 mins)

VI. PyC hands-on! Probabilistic interpretable models (~10 mins)

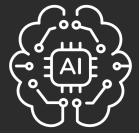
VII. High-level APIs and Conceptarium (~10 mins)

Final Q&A


PyC
AI WITH NOTHING TO HIDE
Out-of-the-box models
(High-level API)
Research in interpretable
architectures
(Mid-level API)
Research in interpretable
modules
(Low-level API)
Out-of-the-Box State-of-the-Art Interpretable Models
Interpretable Probabilistic
Graphical Models
Parallelized (Tensorial)
Probabilistic Inference
Annotations
Annotated
tensors
Interpretable
Layers
Interventions
External
libraries
PyTorch
PyTorch Lightning

High-level API

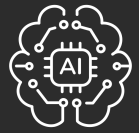
Torch-style



Out-of-the-box Models

```python
# Initialize the CBM
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=concept_annotations,
    variable_distributions=variable_distributions,
    task_names=task_names,
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1}
)
```

High-level API



Out-of-the-box Models

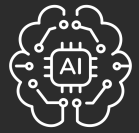
Torch-style

```python
# Initialize the CBM
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=concept_annotations,
    variable_distributions=variable_distributions,
    task_names=task_names,
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1}
)
```

Lightning-style

```python
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=annotations,
    variable_distributions=variable_distributions,
    task_names=['xor'],
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1},
    # Inference engines (both default to DeterministicInference)
    inference=DeterministicInference,
    train_inference=DeterministicInference,
    # Lightning kwargs
    lightning=True,
    loss=loss,
    optim_class=optim,
    optim_kwargs=optim_kwargs
)
```

High-level API



Out-of-the-box Models

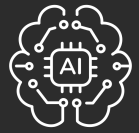
Torch-style

```python
# Initialize the CBM
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=concept_annotations,
    variable_distributions=variable_distributions,
    task_names=task_names,
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1}
)
```

Lightning-style

```python
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=annotations,
    variable_distributions=variable_distributions,
    task_names=['xor'],
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1},
    # Inference engines (both default to DeterministicInference)
    inference=DeterministicInference,
    train_inference=DeterministicInference,
    # Lightning kwargs
    lightning=True,
    loss=loss,
    optim_class=optim,
    optim_kwargs=optim_kwargs
)
```

High-level API



Out-of-the-box Models

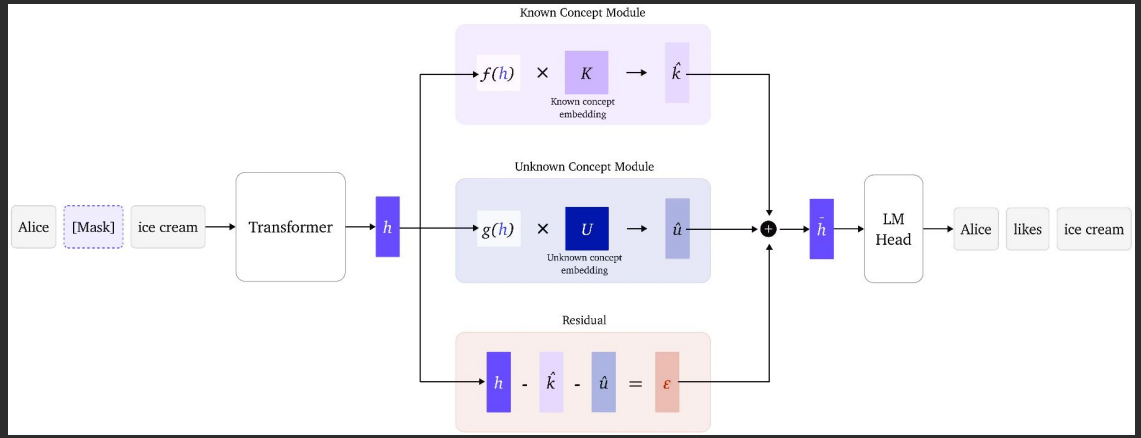
Torch-style

```python
# Initialize the CBM
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=concept_annotations,
    variable_distributions=variable_distributions,
    task_names=task_names,
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1}
)
```

Lightning-style

```python
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=annotations,
    variable_distributions=variable_distributions,
    task_names=['xor'],
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1},
    # Inference engines (both default to DeterministicInference)
    inference=DeterministicInference,
    train_inference=DeterministicInference,
    # Lightning kwargs
    lightning=True,
    loss=loss,
    optim_class=optim,
    optim_kwargs=optim_kwargs
)
```

Steerling-8B

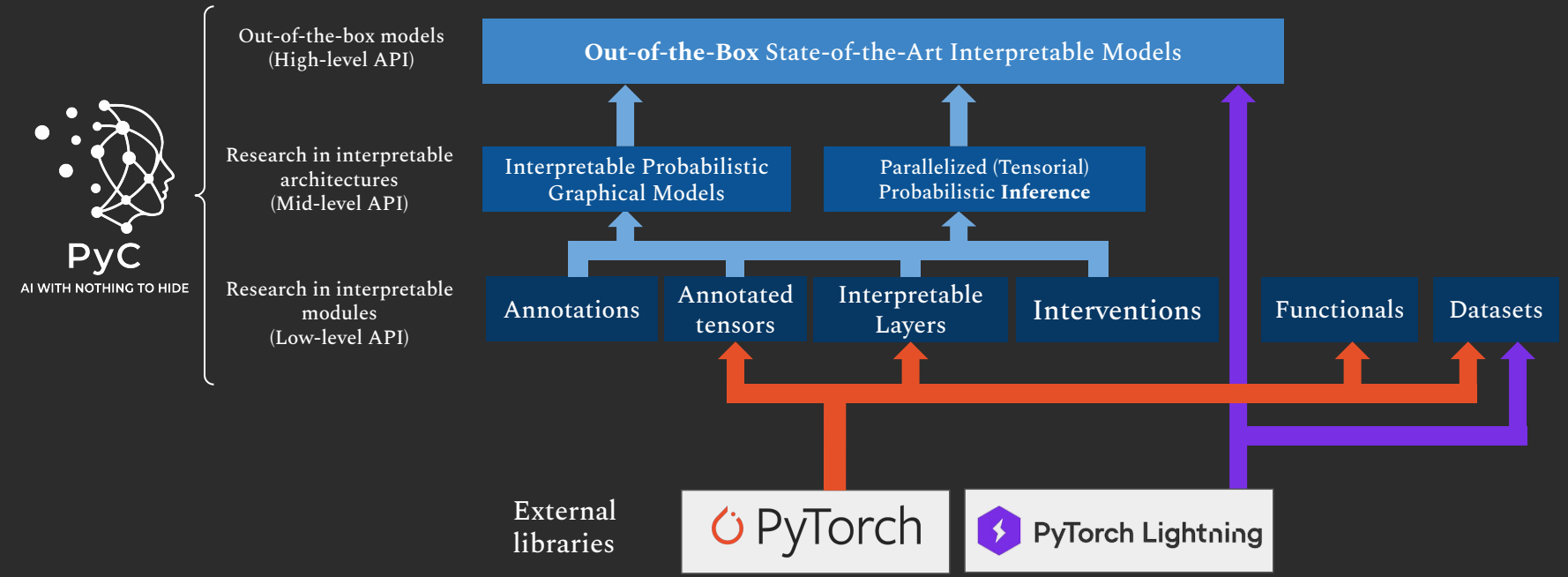



```python
from torch_concepts.steerling import SteerlingModel

steerling_model = SteerlingModel(
    pretrained_components=['backbone', 'known_head', 'unknown_head', 'lm_head'],
    freeze_components=['backbone', 'known_head', 'unknown_head', 'lm_head'],
    use_unknown=True,
    use_epsilon_correction=False,
)
```

Concept-based LLM with 8Bn parameter
by

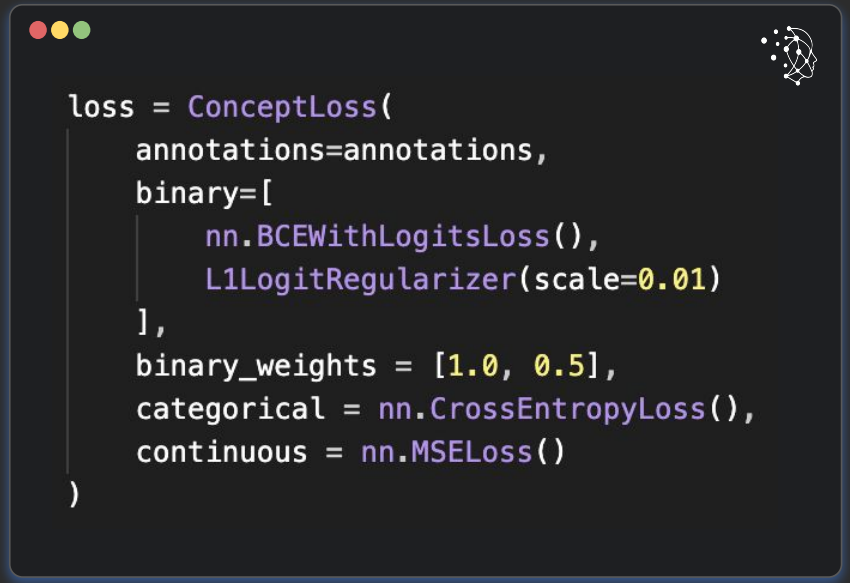
GuideLabs

[1] GuideLabs "Scaling Inherently Interpretable Language Models." Technical report (2026).


PyC
AI WITH NOTHING TO HIDE
Out-of-the-box models
(High-level API)
Research in interpretable
architectures
(Mid-level API)
Research in interpretable
modules
(Low-level API)
Out-of-the-Box State-of-the-Art Interpretable Models
Interpretable Probabilistic
Graphical Models
Parallelized (Tensorial)
Probabilistic Inference
Annotations
Annotated
tensors
Interpretable
Layers
Interventions
Functionals
Datasets
External
libraries
PyTorch
PyTorch Lightning

Loss

Automatic type-aware routing:



```
loss = ConceptLoss(
    annotations=annotations,
    binary=[
        nn.BCEWithLogitsLoss(),
        L1LogitRegularizer(scale=0.01)
    ],
    binary_weights = [1.0, 0.5],
    categorical = nn.CrossEntropyLoss(),
    continuous = nn.MSELoss()
)
```

Loss

Automatic type-aware routing:

```python
loss = ConceptLoss(
    annotations=annotations,
    binary=[
        nn.BCEWithLogitsLoss(),
        L1LogitRegularizer(scale=0.01)
    ],
    binary_weights = [1.0, 0.5],
    categorical = nn.CrossEntropyLoss(),
    continuous = nn.MSELoss()
)
```

Pass it to the model:

```python
# Initialize the CBM
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=annotations,
    variable_distributions=variable_distributions,
    task_names=['xor'],
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1},
    lightning=True,
    loss=loss_fn,
    metrics=metrics,
    optim_class=torch.optim.AdamW,
    optim_kwargs={'lr': 0.02}
)
```

Metrics

Support for:

PyTorch metrics
(concept accuracy, ...)

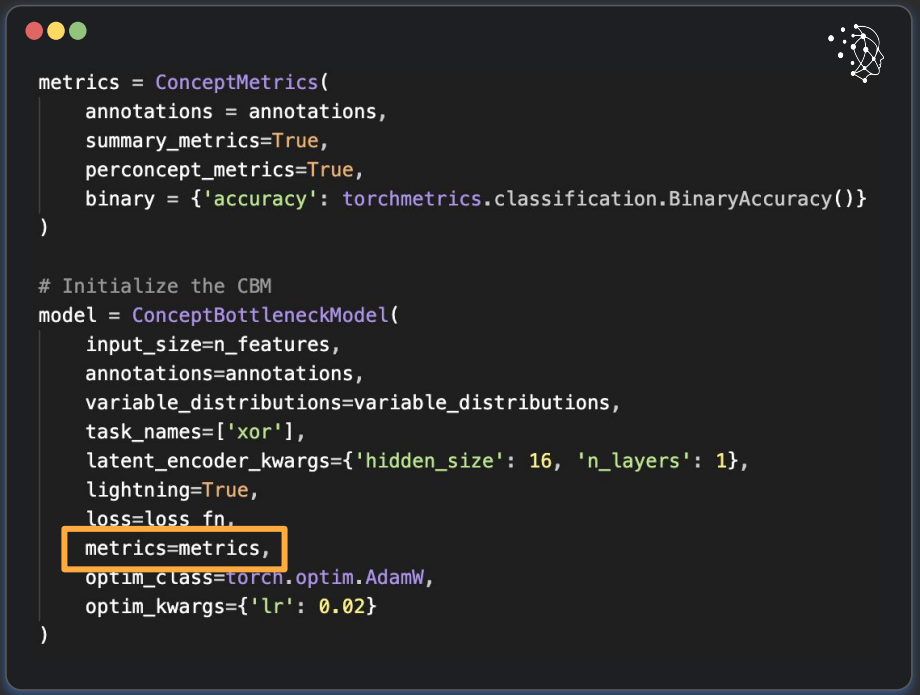
PyC metrics
(interventions, leakage, ...)

Automatic type-aware routing:

```python
metrics = ConceptMetrics(
    annotations=annotations,
    summary=True,     # enable averaged metrics across all concepts of the same type
    per_concept=True, # enable per-concept metrics
    binary={
        # Pre-instantiated
        'accuracy': torchmetrics.classification.BinaryAccuracy(),
        # Class + kwargs
        'f1': (torchmetrics.classification.BinaryF1Score, {'threshold': 0.5}),
        # Class only
        'precision': torchmetrics.classification.BinaryPrecision
    },
    categorical={
        # Class + kwargs
        'accuracy': (torchmetrics.classification.MulticlassAccuracy, {'average': 'micro'}),
        # Class only
        'f1': torchmetrics.classification.MulticlassF1Score
    }
)
```

Metrics

Specify which metrics are tracked during training:



```python
metrics = ConceptMetrics(
    annotations = annotations,
    summary_metrics=True,
    perconcept_metrics=True,
    binary = {'accuracy': torchmetrics.classification.BinaryAccuracy()}
)

# Initialize the CBM
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=annotations,
    variable_distributions=variable_distributions,
    task_names=['xor'],
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1},
    lightning=True,
    loss=loss_fn,
    metrics=metrics,
    optim_class=torch.optim.AdamW,
    optim_kwargs={'lr': 0.02}
)
```

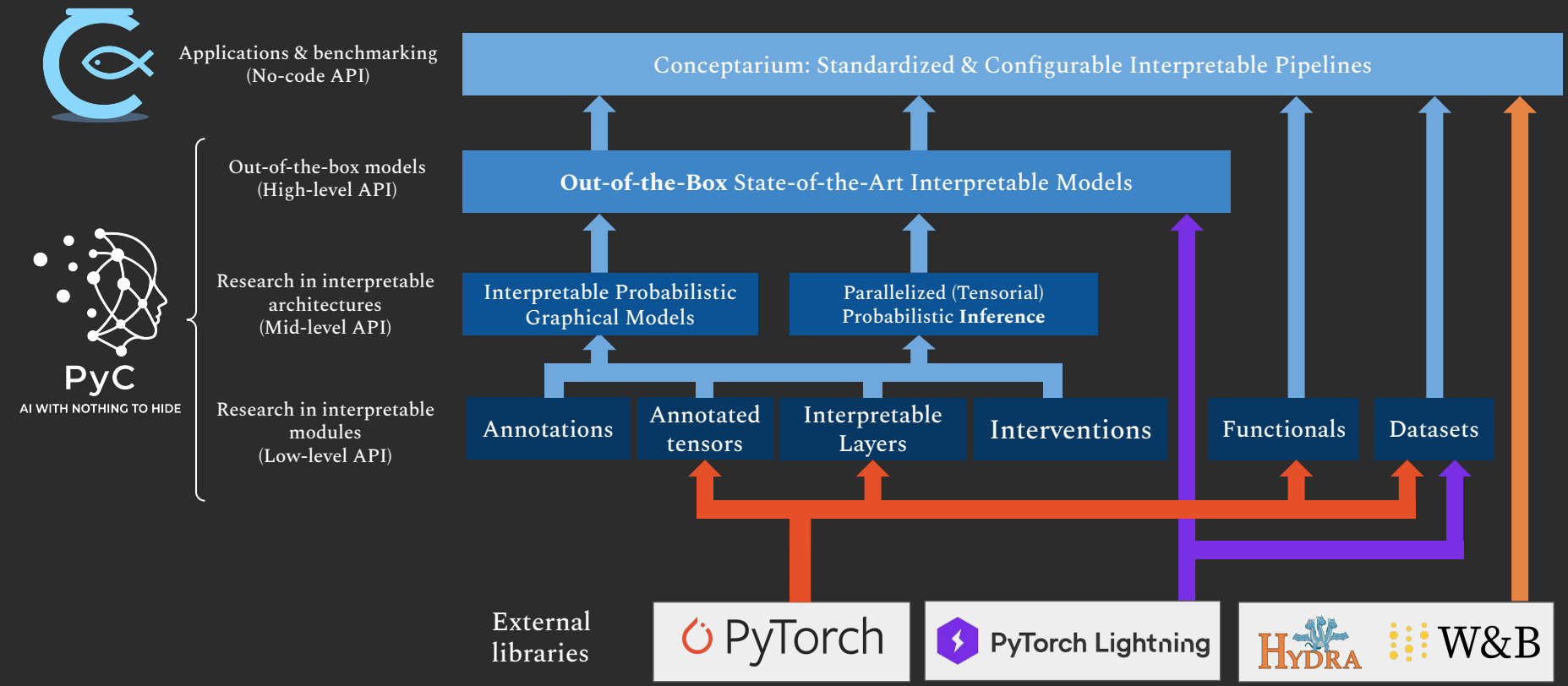
Metrics

Specify which metrics are tracked during training:

```python
metrics = ConceptMetrics(
    annotations = annotations,
    summary_metrics=True,
    perconcept_metrics=True,
    binary = {'accuracy': torchmetrics.classification.BinaryAccuracy()}
)

# Initialize the CBM
model = ConceptBottleneckModel(
    input_size=n_features,
    annotations=annotations,
    variable_distributions=variable_distributions,
    task_names=['xor'],
    latent_encoder_kwargs={'hidden_size': 16, 'n_layers': 1},
    lightning=True,
    loss=loss_fn,
    metrics=metrics,
    optim_class=torch.optim.AdamW,
    optim_kwargs={'lr': 0.02}
)
```

... and which are computed at test:

```python
eval_metrics = ConceptMetrics(
    annotations=annotations,
    summary=True,
    per_concept=True,
    binary={
        'accuracy': torchmetrics.classification.BinaryAccuracy(),
        'f1': torchmetrics.classification.BinaryF1Score()
    }
)
model.eval()
datamodule.setup('test')

test_idxs = datamodule.testset.indices
x_test = dataset.input_data[test_idxs]
c_test = dataset.concepts[test_idxs]

with torch.no_grad():
    out = model(x=x_test, query=concept_names)

eval_metrics.update(preds=out, target=c_test)
print(f"Evaluation results with custom metrics: {eval_metrics.compute()}")
```


Applications & benchmarking
(No-code API)
Out-of-the-box models
(High-level API)
Research in interpretable
architectures
(Mid-level API)
Research in interpretable
modules
(Low-level API)
PyC
AI WITH NOTHING TO HIDE
Conceptarium: Standardized & Configurable Interpretable Pipelines
Out-of-the-Box State-of-the-Art Interpretable Models
Interpretable Probabilistic
Graphical Models
Parallelized (Tensorial)
Probabilistic Inference
Annotations
Annotated
tensors
Interpretable
Layers
Interventions
Functionals
Datasets
External
libraries
PyTorch
PyTorch Lightning
HYDRA
W&B

Conceptarium

All PyC elements can be configured:

- Datasets
- Model
- Losses / metrics
- Training parameters
- etc.

HYDRA

```yaml
defaults:
  - _default
  - _self_

hydra:
  job:
    name: test
  sweeper:
    # standard grid search
    params:
      seed: 1,2,3,4,5
      dataset: celeba, asia, sachs
      model: cbm, cem
      loss: weighted
      loss.concept_weight: 8
      loss.task_weight: 1
```

Main cfg:

```yaml
defaults:
  - _default
  - _self_

hydra:
  job:
    name: test
  sweeper:
    # standard grid search
    params:
      seed: 1,2,3,4,5
      dataset: celeba, asia, sachs
      model: cbm, cem
      loss: weighted
      loss.concept_weight: 8
      loss.task_weight: 1
```

Model cfg:

```yaml
defaults:
  - _commons
  - _self_

# wrapper for 'joint' training mode
_target_: "torch_concepts.nn.ConceptBottleneckModel"

task_names: ${dataset.default_task_names}

inference:
  _target_: "torch_concepts.nn.DeterministicInference"
  _partial_: true
```

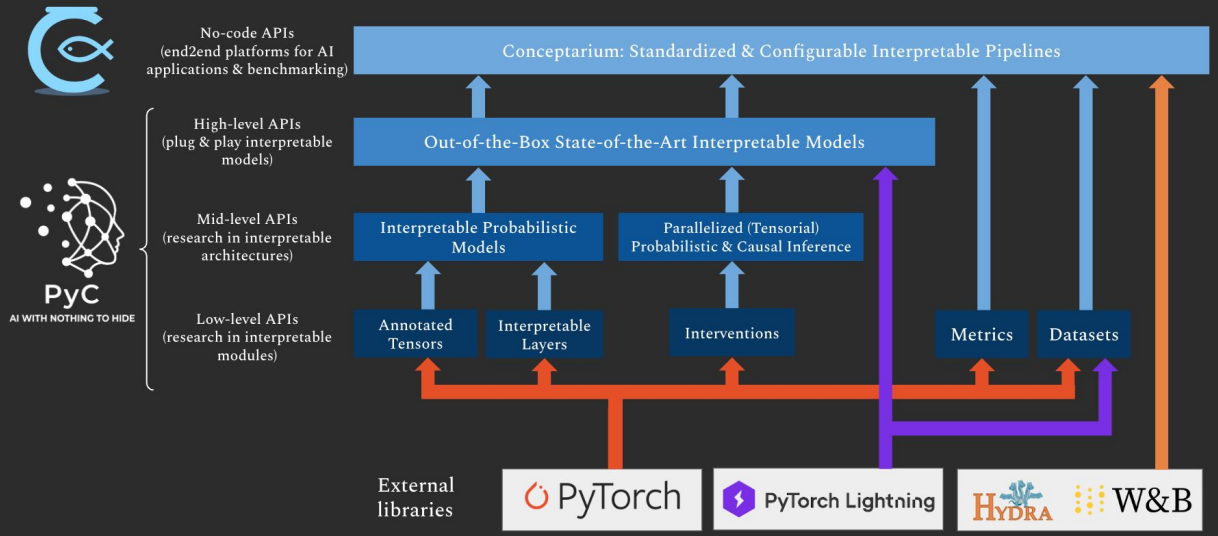
Dataset cfg:

```yaml
defaults:
  - _self_

_target_: torch_concepts.data.datamodules.celeba.CelebADataModule

name: celeba

# backbone handling and embedding precomputation
backbone: facebook/dinov2-base
precompute_embs: true
force_recompute: false

# Task label - which CelebA attribute to predict
default_task_names: [Attractive]

# splitter
splitter:
  _target_: torch_concepts.data.splitters.native.NativeSplitter

# all CelebA attributes are binary facial features
label_descriptions: null
```

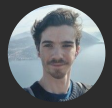
....

And then...

```
CUDA_VISIBLE_DEVICES=1 python conceptarium/run_experiment.py --config-name sweep
```



github link



Giovanni De Felice (PI)

Università della Svizzera Italiana (CH)

Arianna Casanova

University of Liechtenstein (LI)

Gabriele Ciravegna

Intesa Sanpaolo Innovation Center (IT)

David Debot

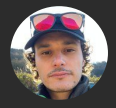
KU Leuven (BE)

Francesco De Santis

Politecnico di Torino (IT)

Mateo Espinosa Zarlenga

University of Oxford (UK)



Edoardo Gabrielli

Sapienza University of Rome (IT)

Tuomas Oikarinen

UC San Diego (US)

What's next?

Standardize research in interpretability

Theory

- Definitions
- Research method

Code

- Model design
- Benchmarks

Methods

- Models
- Metrics

1 → 2 → 3