A Pattern Language for Machine Learning Tasks

Anonymous authors
Paper under double-blind review

Abstract

We formalise the essential data of objective functions as equality constraints on composites of learners. We call these constraints "tasks", and we investigate the idealised view that such tasks determine model behaviours. We develop a flowchart-like graphical mathematics for tasks that allows us to; offer a unified perspective of approaches in machine learning across domains; design and optimise desired behaviours model-agnostically; and import insights from theoretical computer science into practical machine learning. As preliminary experimental validation of our theoretical framework, we exhibit and implement a novel "manipulation" task that minimally edits input data to have a desired attribute. Our model-agnostic approach achieves this end-to-end, and without the need for custom architectures, adversarial training, random sampling, or interventions on the data, hence enabling capable, small-scale, and training-stable models.

1 Introduction

The primary instrument for shaping the behaviour of machine learning (ML) systems is the objective function. We model objectives through a pattern language in which the essential data of a loss is re-expressed as an equational constraint between typed learners. Let Θ_e, Θ_d be the parameter spaces of a model `enc` and `dec`. The reconstruction loss for an autoencoder can then be written (for $\theta_e \in \Theta_e, \theta_d \in \Theta_d$) as

$$\underset{\theta_e,\theta_d}{\arg\min} \left(\mathbb{E}_{x\sim\mathcal{D}}\left[\mathbf{D}\left(\mathtt{dec}_{\theta_d}(\mathtt{enc}_{\theta_e}(x)), x\right)\right]\right),$$

where we take the expected value over a data distribution $\mathcal{D}$ on a feature space $\mathcal{X}$ of a statistical divergence $\mathbf{D}$ (such as cross-entropy or log-likelihood) between two expressions that, under ideal conditions, should be equal: the decoding of the encoding of some data x, and the original x. In our language, we focus on the two expressions that want to be equal, and we call this equational constraint a *task*.

In practice, designing a good objective function incorporates many technical choices, such as the architecture, the statistical divergence, and the choice of training data (Ciampiconi et al., 2023; Richardson, 2022; Terven et al., 2023). However, these choices are often made by heuristics, or rationalised post hoc. While such choices are sometimes required to make training tractable, they are not always relevant to understanding the final behaviour of the trained model. Instead, we propose and investigate the idealised view that *tasks determine model behaviour*.

To explore the expressive power of tasks, we abstract away implementation details such as architecture and training by idealising models to be universal function approximators that can, in principle, perfectly optimise objective functions: *in other words, we consider learners to be typed function-variables*. Thus each task can be viewed purely as an equational constraint on the behaviour of the learners, exactly analogous to equational constraints on the possible values of variables in algebra (Definition 2.6).

Unpacking this bird's eye perspective of deep learning for a mathematical audience, the view we propose is that deep learning is nonlinear representation theory: instead of linear maps representing groups, smooth maps represent higher algebraic structures such as coloured PROPs (Yau, 2008) for typed function composition, and the role of gradient descent is to convert data and compute into instances of these functorial representations. Unpacking the same perspective as a value proposition for practitioners, we suggest that deep learning

solutions may be constructed declaratively, by writing down task equations that capture what we want them to do in terms we understand, letting neural networks and gradient descent figure out how to satisfy those equations by themselves.

We represent these task equations using string diagrams, which are a natural and recurring syntactic choice in mathematical domains concerned with formal reasoning about the behaviour of processes composed sequentially and in parallel (Selinger, 2011); for us, these diagrams expose how multiple objectives interact across composed learners, which is precisely the regime in which informal engineering choices become brittle. We develop a *pattern language* with these diagrams, grouping together objectives behaviourally, and visually simplifying formal reasoning with their higher-algebraic compositional structure as topological manipulation of string diagrams.

Our overarching aim is to provide theoretical foundations and an expressive language that bridges existing methods and enables rigorous reasoning, supported by practical demonstrations designed to illustrate rather than benchmark empirically. The theory we present is deliberately broad and architecture-agnostic, and the examples we provide are designed to provoke and inspire rather than exhaustively demonstrate. Large-scale empirical validations or competitive benchmarking against established architectures, though important for broader practical adoption, fall outside the immediate scope of this paper.

1.1 Contributions

We make three primary contributions in this work. Our first, theoretical contribution is the formalisation of a common but informal standard procedure in deep learning, which may be summarised as the following recipe: Characterise desired behavior via equational constraints (tasks) between learners → Implement tasks by treating neural networks as universal approximators → Convert equations to loss function by a choice of hyperparameters, namely weighted sum of constituent losses and choice of divergences.

Recipe: from specification to objective

- Specify desired behaviours diagrammatically before committing to architectures.
- Compile tasks to objectives via weighted divergences on the codomains involved.
- Refactor competing designs algebraically (using specialisation/refinement) to compare guarantees.
- Train and assess using the same task equations, yielding contract-aligned tests by construction.

Our second, theoretical contribution is a demonstration that within our framework, we may analyse, predict, and design behaviours of models. By diagrammatic algebraic reasoning, we can analyse the behaviour of complex models via synthetic relationships between tasks we call *specialisation* (Definition 2.8) and *refinement* (Definition 2.14), grounded by well-understood tasks we call *patterns* (Section 2.4). Altogether, we may use these techniques to understand and compare the behaviour of models before committing to potentially costly training. Example 2.12 and Propositions 2.17,3.6 and 3.8 illustrate the kinds of formal reasoning our language enables.

Our third, practical contribution and proof-of-concept is the implementation of a novel task we call `manipulation` (Section 3), which formalises (Bancilhon & Spyratos, 1981) the problem of viewing and editing a targeted attribute of data while "leaving other aspects the same". As examples in image domains, we change only the colour of a shape (Figure 3); change the value of a handwritten digit without affecting other stylistic properties (Figure 4); and change only whether a person is smiling in an image (Figure 8). Even in these toy domains, we observe a range of benefits we expect to scale: by following our recipe we obtain architecture-agnostic (Table 1) style-transference models without the need for randomness, adversarial training, or modality- and architecture-specific interventions, with good interpolation properties (Figure 8).

1.2 Positioning within the literature

Decision-theoretic ML. Classical risk-minimisation views such as ERM and MLE treat objectives as scalar summaries of performance (Vapnik, 1998), while optimisation-centric Bayesian perspectives (Knoblauch

An applied example of specification to objective

1. Start with a natural-language requirement, e.g., "recolour a shape without altering anything else". Identify the observable attributes (colour) and the parts to preserve (all other image properties).
2. Introduce learners that witness the behaviour: a reader `get` for the attribute, a writer `put` that edits, plus auxiliary encoders/decoders when convenient.
3. Write the desired behaviour as equations (Classify, PutGet, GetPut, Undoability; see Task 3.1) so that each equality captures a promise we want the system to satisfy.
4. Compile each equation into a loss term by choosing divergences and weights, producing an implementable optimisation objective.
5. Train the learners with standard optimisers and re-use the same equations as diagnostics: measure attribute flip accuracy with Classify, and check non-target preservation with GetPut/Undoability.

et al., 2022) analyse posterior design; marketplace-style multi-model frameworks such as Machine Learning Markets (Storkey et al., 2011) provide yet another lens on how interacting objectives equilibrate. Our task language complements these approaches by reasoning about how multiple such objectives compose and constrain one another across interacting learners.

Graphical models and probabilistic programming. Plate diagrams and graphical models capture conditional independences and factorisations (Koller & Friedman, 2009). By contrast, our wires track typed computational flows with learnable morphisms, making them better suited to multi-loss design questions rather than inference over fixed structures; we point to probabilistic programming when stochastic semantics are the focus.

Neurosymbolic and compositional modules. Neurosymbolic systems (d'Avila Garcez & Lamb, 2020) often pair neural back-ends with symbolic engines or constraints. Our calculus operates one abstraction layer above, equipping designers with equational templates for objectives that can then be instantiated with neural modules, rather than hard-coding symbolic reasoning.

Bidirectional transformations and lenses. The manipulation case study imports laws from bidirectional programming and view-update research (Abou-Saleh et al., 2018; Nakano, 2021). The diagrammatic setting lets us treat these laws as ready-made task equations, preserving their provenance while compiling them directly into ML optimisation objectives.

Adjacent style-transfer and disentanglement methods. The manipulation case study is closely related to style-transfer pipelines such as CycleGAN (Zhu et al., 2017) and disentanglement frameworks like β-VAE (Higgins et al., 2017), which pursue related goals but with different guarantees: the former emphasise distributional alignment, the latter global latent factorisation. Our manipulation task targets local, per-instance attribute control with explicit behavioural contracts; Section 3 clarifies these distinctions formally.

2 Tasks and patterns

2.1 Notation

String diagrams provide a typed, topological syntax for wiring learners; sequential composition corresponds to placing boxes left-to-right, and parallel composition stacks wires vertically. We use the generators in Figure 1 throughout the paper.

In terms of symbolic notation, we use $\mathcal{X}, \mathcal{Y}$ for feature spaces and domains that serve as carriers for inputs and outputs of learners; X, Y for random variables on feature spaces used in expectations and probabilistic reasoning; $\mathcal{D}$ for data distributions on $\mathcal{X}$ that supply samples for training and evaluation; Θ for parameter spaces that specify learnable families (e.g., encoders/decoders); $\mathbf{D}$ for divergences or losses that compile task equations into optimisation objectives; and α, β for hyperparametric constants used in weighted sums of

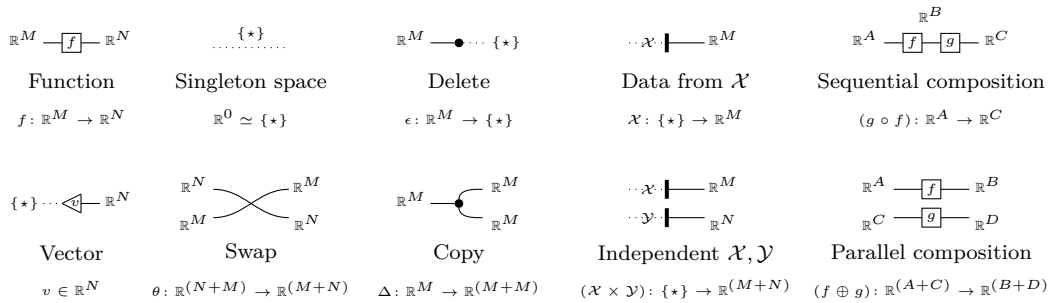


Figure 1: Wires carry typed data, black boxes are learners, white boxes are fixed functions. Vectors are depicted as triangular nodes with only output wires, reminiscent of bra–ket notation; (co)associative (co)monoids such as copy/delete and add/zero appear as circular nodes. Encoders and decoders may be drawn as trapezia, and distributions as thick bars. This grammar is sufficient for reading every diagrammatic derivation in the sequel.

objectives. We use `monospace` to refer to patterns and their constituent learners, and SMALL CAPS to refer to the constituent task equations of patterns for ease of reference when reasoning. Additionally, we use colours (like this or this) to distinguish different types of data or components in diagrams.

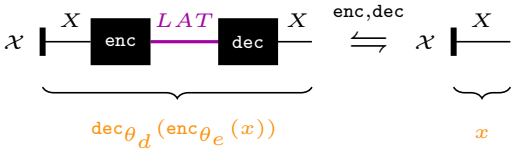
2.2 An introduction via worked examples

Rather than starting with abstract definitions, we will begin gently with a series of worked examples to introduce the core ideas of our framework in action, and to make the reader comfortable with string-diagrammatic representation and reasoning. While the formal details (Appendix A) involve category theory, the power of string diagrams lies in their intuitive visual nature: by reading the diagrams as flowcharts from left to right, practitioners can leverage these diagrams to reason about ML tasks without needing to fully grasp the underlying mathematical formalism.

As a running example, we present the reconstruction loss of autoencoders once again.

Example 2.1. For a hyperparametric choice of divergence $\mathbf{D}$, where X is an input datatype, and LAT is the datatype of the latent space, the two components of the empirical risk minimisation of the autoencoder task consist of (1) applying the encoder `enc` (typed $X \to LAT$) followed by the decoder `dec` (typed $LAT \to X$) to inputs x drawn from a source $\mathcal{D}_X$ of data over the datatype X, *which should be equal to* (2) the original x. Recall that by the Universal Approximation Theorem, we seek to treat learners as function-variables. So we abstract away the parameter spaces θ, and simply depict the learners as black-boxes.

$$\mathbb{E}_{x \sim \mathcal{X}}\left[\mathbf{D}\Big(\underbrace{\mathtt{dec}_{\theta_d}(\mathtt{enc}_{\theta_e}(x))}_{(1)}, \underbrace{x}_{(2)}\Big)\right]$$

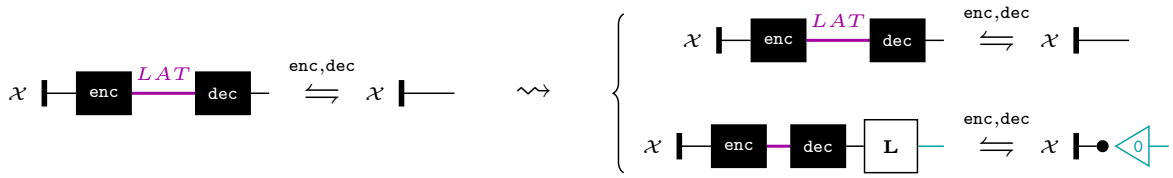


The diagrammatic notation is formally equivalent to the traditional symbolic notation with the addition of type-information about inputs and outputs. In summary, nodes depicted as various shapes are functions, and wires are datatypes which can be understood as carrying information.

We observe that the function-variable perspective immediately suggests two ways of *specialising* such tasks viewed as equational objectives: by imposing additional tasks, or by specifying architectural choices.

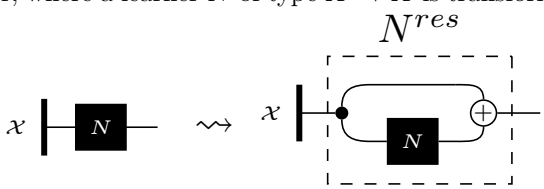
Example 2.2 (Perceptual losses as an additional objective)**.** Adding a perceptual loss $\mathbf{L} : X \to \mathbb{R}^{\geq 0}$ to an autoencoder is a common example of task specialisation by additional objectives. Adding additional objectives means that the learners should optimise the original tasks as well as the new ones. In practice, multiple tasks may be combined into a single objective function by means of weighted summation with positive hyperparametric coefficients α, β, but we abstract away such choices in our visual presentation to

leave only the equational data. Depicting processes with no learnable parameters as white boxes, we have:



$$\mathrm{argmin}_{\theta_e,\theta_d}(\mathbb{E}_{x\sim\mathcal{X}}[\mathbf{D}(\mathtt{dec}_{\theta_d}(\mathtt{enc}_{\theta_e}(x)),x)]) \rightsquigarrow \mathrm{argmin}_{\theta_e,\theta_d}(\alpha\mathbb{E}_{x\sim\mathcal{X}}[\mathbf{D}(\mathtt{dec}_{\theta_d}(\mathtt{enc}_{\theta_e}(x)),x)] + \beta\mathbb{E}_{x\sim\mathcal{X}}[\mathbf{D}(\mathbf{L}(\mathtt{dec}_{\theta_d}(\mathtt{enc}_{\theta_e}(x)),0)])$$

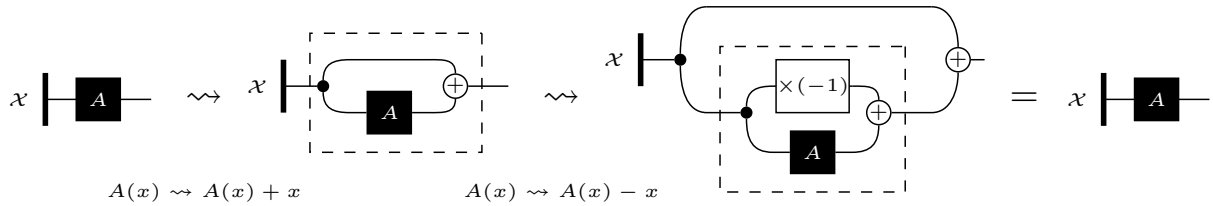
Example 2.3 (Residuation as an architectural choice)**.** A common example of an architectural specialisation is adding a residual to a learner, where a learner N of type $X \to X$ is transformed into $N^{res} := x \mapsto N(x)+x$, depicted as:



In an exact analogy with symbolic variable substitution, specialisation in this manner diagrammatically rewrites a function-variable in place as another diagram with matching input-output type constraints (which may itself contain a function-variable). The full formal semantics for such substitutions are provided in Appendix A.2. Intuitively, since a universal function approximator can be any function, it can be a particular function (or subset of functions) of the same input-output type by a rewrite. Since such architectural choices generally correspond to imposing inductive biases, these rewrites are usually one-way.

Although our framework is too abstract in its current formulation to remark directly on properties during training, we may still reason about the relative expressivity of learners by specialisation alone, which we later expand to compare different tasks (Definition 2.14).

Example 2.4 (Residuations do not alter expressivity)**.** In the presence of negation, a nonresiduated learner A can learn anything its residuated version can, and vice versa, as the following cycle of specialisations demonstrates:



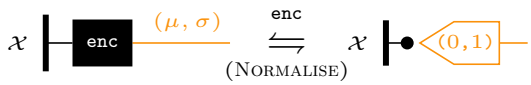
So, specialisations allow us to analyse and compare the behaviour of models in terms of their constituent tasks. More productively, specialisations also give us a foothold for explaining and constructing model variants. As a demonstrative example, we may obtain a variational autoencoder (VAE) (Kingma & Welling, 2022) from a regular autoencoder by the two kinds of specialisation we have described.

Example 2.5 (`VAE`)**.** Suppose we wish to construct a probabilistic autoencoder. As a first step, we may achieve this by specialising the output type of the encoder to be a space (mean, variance) of parameters for Gaussians, and specialising the decoder to be internally structured as the sequential composite of sampling from a Gaussian of the input (mean, variance) followed by a learner.

By inspection, there are two salient cases in which models trained according to the above task may degenerate into deterministic behaviour. The first is when the encoder exclusively chooses very small variance encodings,

and the second is when the encoder chooses means μ that are very far apart: in both cases, the decoder receives point-like encodings.

Both cases are addressed by specialisation with an additional normalisation task which requires the outputs of the encoder to be close to the parameters of the unit Gaussian $(0, 1)$.



Surprisingly, this behaviourally declarative presentation of VAEs is equivalent to the traditional probabilistic form when the statistical divergence is KL (Rocca, 2021).

Altogether, these examples illustrate how tasks give a foothold for inspecting and designing desired behaviours, and how specialisations may be used to pre-empt unwanted behaviours. We further elaborate on the use of this form of reasoning for informing task design in Appendix B, in the context of obtaining guarantees for the `manipulation` task.

2.3 Tasks, formally

We assume the following contextual data, omitted if there is no confusion. Let X, Y denote datatypes; Σ a set of processes f, each of which has (possibly empty) learnable parameter datatypes $\mathfrak{p}_f$. An atomic task is a process-theoretic equational constraint on learners specifying that f should behave like g on all inputs. The objective function of an atomic task φ of type $X \to Y$ equipped with distribution $\mathcal{D}_X$ corresponds to a map $\mathfrak{p}_\varphi \to \mathbb{R}$ that sends $\pi \mapsto \mathbb{E}_{x \sim \mathcal{D}_X}\left[\mathbf{D}\big(\mathrm{sys}_{\varphi;\pi}(x), \mathrm{spec}_{\varphi;\pi}(x)\big)\right]$ for some choice of statistical divergence $\mathbf{D}$. A learner can optimise multiple atomic tasks simultaneously by optimising a combination α of the atomic objective functions (commonly obtained by taking a weighted sum).

Definition 2.6 (Tasks)**.** An *atomic task* φ is a tuple $(f, g, \mathcal{D}_X, \mathfrak{p})$, where $f, g : X \to Y$ are composite processes of Σ, $\mathcal{D}_X$ is a distribution over X, and $\mathfrak{p} \subseteq \mathfrak{p}_f \oplus \mathfrak{p}_g$ is a space of trainable parameters. We indicate the *system* f and *specification* g as sys_φ and spec_φ and similarly the *domain* X and *codomain* Y as $\mathrm{dom}(\varphi)$ and $\mathrm{cod}(\varphi)$. A *compound task* Φ (or just *task*) is a non-empty set of tasks. As we have seen in previous examples, we notate a task Φ as a collection of atomic tasks $\mathrm{sys}_\varphi \leftrightharpoons \mathrm{spec}_\varphi$, where superscripts on the harpoons indicate which learnable parameters are governed by each atomic task.

Tasks become concrete objective functions by a hyperparametric choice of divergences for atomic tasks, followed by a combination via a weighted sum with hyperparameter coefficients, or more generally a *compound function*. As such, a particular objective function that instantiates a task is one where the choices for the measure of statistical divergence and compound function have been made. Beyond these hyperparameters, tasks and objective functions may be viewed as informationally equivalent.

Definition 2.7 (Objective function)**.** Let $\Phi = \{(f_i, g_i, \mathcal{D}_i, \mathfrak{p}_i)\}_{i \in N}$ be a compound task with N atomic tasks. Let $l \in \Sigma$ be a learner of Φ. An *objective function* for l is a tuple $(\Phi_l, \mathcal{D}, \alpha)$ where $\Phi_l = \{(f_i, g_i, \mathcal{D}_i, \mathfrak{p}_i) \mid \mathrm{para}(l) \subseteq \mathfrak{p}_i\}$ is the set of all tasks on which l is optimised, $\mathcal{D}$ is a set of *statistical divergences* $\mathbf{D}_{(\varphi \in \Phi_l)} : \mathrm{cod}(\varphi) \times \mathrm{cod}(\varphi) \to \mathbb{R}^{\geq 0}$, and the *compound function* α is a function $(\mathbb{R}^{\geq 0})^{\times |\Phi_l|} \to \mathbb{R}^{\geq 0}$ that is differentiable, and typically non-decreasing in each argument.

Definition 2.8 (Specialisation)**.** A *specialisation* of a task Φ is a task obtained by imposing additional constraints on Φ. We identify two primary forms, both maintaining or strengthening original behavioural guarantees, with architectural specialisation additionally enabling explicit engineering of inductive biases into models.

- *Objective Specialisation* is adding further tasks, represented formally as extending the task set: $\Phi_{\mathrm{spec}} = \Phi \cup \{\psi_i\}_{i \in I}$, where each ψ_i is an additional atomic or compound task imposing further behavioural constraints on learners.
- *Architectural Specialisation:* Let Φ contain an atomic task involving a learner $f \in \Sigma$ with type $X \to Y$. A task Φ' is an architectural specialisation of Φ if f is diagrammatically substituted with

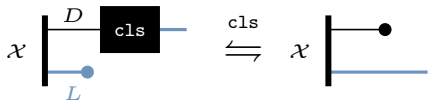
a composite process $C[f] : X \to Y$ such that $C[f] \in \Sigma^*$ and $\mathrm{dom}(C[f]) = X$, $\mathrm{cod}(C[f]) = Y$. Here, $C[f]$ is a well-typed composite in the free symmetric monoidal category over Σ, constructed by replacing f with a structured process diagram using additional (possibly parameterised) components, while preserving type compatibility. The substitution is interpreted as a one-way rewrite rule on diagrams: $f \rightsquigarrow C[f]$, valid in the universal approximator regime (Appendix A.2).

Throughout the following sections, we present a selection of tasks and relate them to their respective objective functions, organised by learning paradigm. The following examples are not exhaustive, but serve to cast familiar ML regimes in diagrammatic form, helping readers build intuitions about the notation. We illustrate `classification` and `regression` [Pattern 2.9] for supervised learning; `autoencoding` [Pattern 2.10] and `energy minimisation` [Pattern 2.13] for unsupervised learning; `GAN` [Pattern 2.11] and `VAE` [Example 2.5] for generative modelling; and `CycleGAN` [Example 2.12] for transfer learning.

2.4 Patterns are "nice" tasks

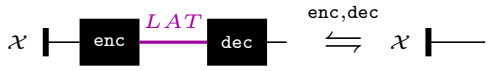
On the view that behaviour can be captured equationally, it would be desirable to have a bank of correspondences between tasks and their behaviours. We do not expect there to be a general and systematic method to translate between natural-language behavioural specifications and tasks; if such a method existed, then all of deep learning would be reduced to hyperparameter search. There are often many different ways to approach a problem in ML, much like there is no single correct way to write software, or design a building. This suggests to us the view of *patterns*: some tasks are well understood, usable modularly and in many contexts, and easily modifiable, and such tasks can be viewed as *design patterns* – borrowing a term from software engineering (Beck & Cunningham, 1987)[1]. In this section, we suggest some examples of patterns that correspond to well-studied methods and paradigms in ML, and we show how to use patterns as an accessible basis to analyse models.

Pattern 2.9 (`classification`)**.** Given a data-label pair (d, l) drawn from $\mathcal{D}_X$ with labels $l \in L$, a *classifier* $\mathtt{cls} : D \to L$ is a function that solves the `classification` task, in which it seeks to reconstruct the label from the data:



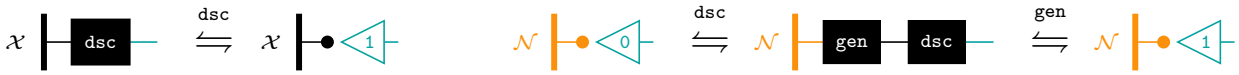
This can be done by minimising the corresponding objective function $\mathbb{E}_{(d,l)\sim\mathcal{D}_X}[\mathbf{D}(\mathtt{cls}(d), l)]$ for some measure of statistical divergence $\mathbf{D}$ on the label space, which may be continuous to encompass regression.

Pattern 2.10 (`autoencoding`)**.** As we have seen, given a data distribution $\mathcal{D}_X$ over X and some latent space LAT, an *autoencoder* consists of an *encoder* $\mathtt{enc} : X \to LAT$ and a *decoder* $\mathtt{dec} : LAT \to X$ which cooperate to reconstruct the identity over the observed distribution:



This corresponds to minimising $\mathbb{E}_{x\sim\mathcal{D}_X}[\mathbf{D}(\mathtt{dec}(\mathtt{enc}(x)), x)]$.

Pattern 2.11 (`GAN`)**.** Given a data distribution $\mathcal{D}_X$ over X and noise distribution $\mathcal{N}$ over N, a *generative adversarial network* (`GAN`) consists of a *generator* $\mathtt{gen} : N \to X$ and a *discriminator* $\mathtt{dsc} : X \to [0, 1]$. The prosaic explanation that "the discriminator seeks to distinguish real data from fake data while the generator aims to fool the discriminator" translates directly into a task description: where 1 indicates "real" and 0 indicates "fake":



In other words, the discriminator `dsc` seeks to minimise some positive combination of the terms $\mathbb{E}_{x\sim\mathcal{D}_X}[\mathbf{D}(\mathtt{dsc}(x), 1)]$ and $\mathbb{E}_{x\sim\mathcal{D}_X}[\mathbf{D}(\mathtt{dsc}(\mathtt{gen}(x)), 0)]$, while the generator `gen` seeks to minimise $\mathbb{E}_{x\sim\mathcal{D}_X}[\mathbf{D}(\mathtt{dsc}(\mathtt{gen}(x)), 1)]$.

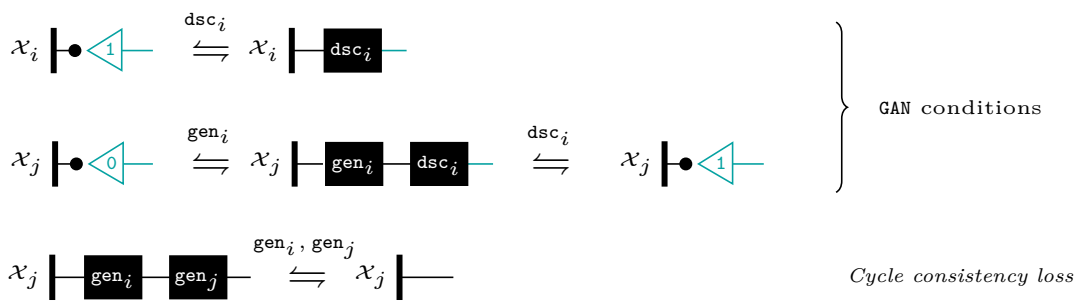
[1] And before that, architecture and urban design (Alexander et al., 1977).

For the sake of compactness, on the right hand side, we compressed the two adversarial tasks into one figure where the discriminator and generator pull into opposite directions.

2.5 Analysing complex tasks

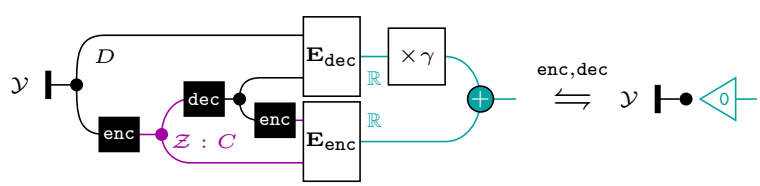
We can analyse the intended functions of models by viewing them as composites of simple patterns. We have already seen in Example 2.5 how a VAE is analysable by inspection as a specialised regular autoencoder. As a second example, a `CycleGAN` is two `GAN`s on different distributions, whose generators are mutually autoencoders by a *cycle consistency loss* (Zhu et al., 2017). This suggests that the generators encode the distributions into each other in a reversible manner, and indeed this kind of style transfer between distributions is what a `CycleGAN` does in practice.

Example 2.12 (`CycleGAN`)**.** Below, i, j are nonequal indices taking values in $\{0, 1\}$, where $\mathcal{D}_0$ and $\mathcal{D}_1$ are different distributions on the same space X: typically these are two classes of images.



We can also upgrade informal intuitions into formal derivations. For example, on the account of Ranzato et al. (2007), a broad class of unsupervised learning techniques — including PCA and k-means — are specialisations of the `energy minimisation` task, which may be considered an `autoencoding` task from a latent "code" space subject to the representations minimising a measure of "energy". We can in fact formalise the relationship between these forms of unsupervised learning and `autoencoding`, showing that under mild assumptions, they are the same in the computational limit.

Pattern 2.13 (`energy minimisation`)**.** `energy minimisation` consists of; three types of systems: D(ata), C(ode), and $\mathbb{R}^{\geq 0}$; two learnable processes: an encoder $\mathtt{enc} : D \to C$ and a decoder $\mathtt{dec} : C \to D$; two user-supplied *energy functions*: $\mathbf{E}_{\mathtt{enc}} : C \times C \to \mathbb{R}^{\geq 0}$ and $\mathbf{E}_{\mathtt{dec}} : D \times D \to \mathbb{R}^{\geq 0}$; and a user-supplied constant $\gamma \in \mathbb{R}^{\geq 0}$. Provided a distribution of inputs $\mathcal{Y}$ on D, and a distribution $\mathcal{Z}$ on C the system seeks to minimise $\mathbb{E}_{y \sim \mathcal{Y}, z \sim \mathcal{Z}}[\gamma \mathbf{E}_{\mathtt{enc}}(\mathtt{enc}(y), z) + \mathbf{E}_{\mathtt{dec}}(y, d(z))]$. Such a task is called *code-extracting* when $\mathcal{Z} = \mathtt{enc}(\mathcal{Y})$, in which case, diagrammatically, this corresponds to the following task:

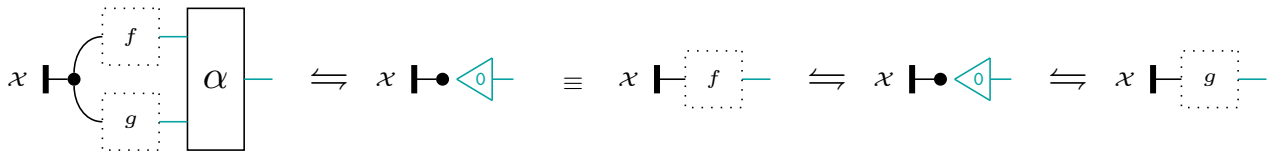


To formally relate code-extracting `energy minimisation` and `autoencoding`, we introduce a relationship between tasks called *refinement*, which states that perfectly solving Φ allows one to construct perfect solutions for Ψ. When Φ and Ψ refine each other, the tasks are "the same in the computational limit"; a perfect autoencoder is a perfect code-extracting energy minimiser, and vice versa. These relationships are a proxy for the behaviour and relative power of concrete implementations of tasks.

Definition 2.14 (Refinement and equivalence of tasks)**.** Task Φ *refines* task Ψ if, by treating the atomic tasks as equations, the processes of Φ may be composed to satisfy the equations of Ψ. Φ and Ψ are *equivalent* if they refine one another, which we denote $\Phi \equiv \Psi$.

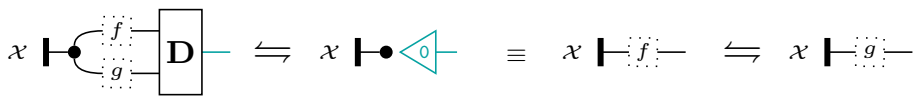
Now we aim to prove that autoencoders and energy-minimisers are identical in the computational limit: that a perfect deterministic autoencoder yields a perfect energy-minimiser, and vice versa.

Lemma 2.15. For all well-typed f, g, and for any positive linear combination $\alpha : \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$:



Proof. For the forward refinement, as α is a positive linear combination, we have for all $x \in \mathcal{D}_X$ that $\alpha(f(x), g(x)) = \alpha_1 \cdot f(x) + \alpha_2 \cdot g(x) = 0$. If f and g are constant-functions 0, we are done. Otherwise, positivity implies that $\alpha_1 \cdot f(x) = \alpha_2 \cdot g(x) = 0$, and since $\alpha_1, \alpha_2 \in \mathbb{R}^{\geq 0}$, then $f(x) = 0 = g(x)$, which is the desired task. For the backwards refinement, if $f(x) = 0 = g(x)$ for all $x \in \mathcal{D}_X$, then $\alpha(f(x), g(x)) = \alpha_1 \cdot f(x) + \alpha_2 \cdot g(x) = 0$. □

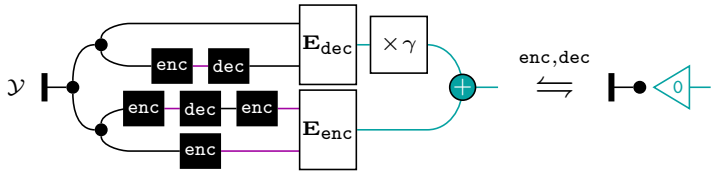
Lemma 2.16. For a real-valued pairwise measure $\mathbf{D} : \mathcal{D}(Y) \times \mathcal{D}(Y) \to \mathbb{R}^{\geq 0}$ on the space of distributions over Y, the positivity axiom $\mathbf{D}(\mathcal{Y}_1, \mathcal{Y}_2) = 0 \iff \mathcal{Y}_1 = \mathcal{Y}_2$ implies, for (almost[2]) all $f, g : X \to Y$ and $\mathcal{D}_X$:



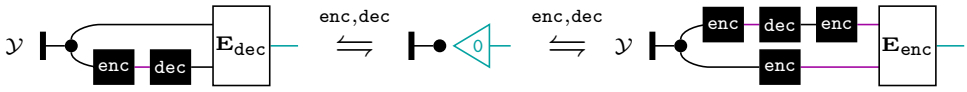
Proof. For the forward refinement, we assume that $\mathbf{D}(f(\mathcal{D}_X), g(\mathcal{D}_X)) = 0$. By positivity of $\mathbf{D}$, $f(\mathcal{D}_X) = g(\mathcal{D}_X)$, which is the right hand task. For the backward refinement, if $f(\mathcal{D}_X) = g(\mathcal{D}_X)$, then, by positivity, $\mathbf{D}(f(\mathcal{D}_X), g(\mathcal{D}_X)) = 0$. □

Proposition 2.17. If `enc` and `dec` are deterministic and $\mathbf{E}_{\texttt{enc}}$ and $\mathbf{E}_{\texttt{dec}}$ are positive (e.g. metrics or statistical divergences), then `energy minimisation` $\equiv$ `autoencoding`.

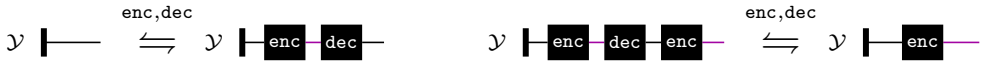
Proof. As `enc` and `dec` are functions, we may copy them through their outputs to re-express `energy minimisation` as:



By Lemma 2.15, this is equivalent to:



Recall that $\mathbf{E}_{\texttt{enc}}$ and $\mathbf{E}_{\texttt{dec}}$ are positive by definition, hence Lemma 2.16 allows us to express the two minimisations as:



The left task is `autoencoding`, so we have that `energy minimisation` refines `autoencoding`. For the other direction, we observe that `autoencoding` refines the right task by postcomposing both sides with `enc`, and as the left and right tasks together are equivalent to `energy minimisation`, we have the claim. □

[2]When the function space containing f and g is large, the edge case where f and g are nonconstant and $f = -g$ is measure-theoretically negligible. Since we are concerned with behaviour in the computational limit, this is an acceptable assumption for our purposes.

2.6 Proof of concept: Tasks from specifications - the stack

In this section and the following (Section 3), we will provide two examples of utilising the proposed perspective to design new tasks. First, we will propose and experimentally verify a set of tasks that is restrain two interacting learners to perform the *push* and *pop* operations of a `stack`. While by itself the `stack` is not particularly interesting, this section is more used as a first introduction to working with tasks to create desired behaviour. In Section 3 we will introduce a more involved tasks for manipulating attributes in data.

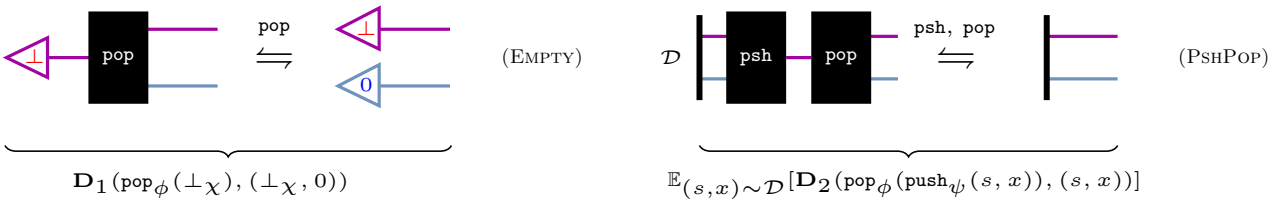
The well-known data structure `stack` consists of two interacting operations: *push* and *pop*.

Task 2.18 (`stack`)**.** Given a data distribution $\mathcal{D}_X$ over X and a pretrained `autoencoder` (`enc`: $X \to \mathbf{LAT}$, `dec`: $\mathbf{LAT} \to X$), a `stack` consists of a *empty stack* $\perp : \star \to S$, a push operation $psh : S \times X \to S$ and a pop operation $pop : S \to S \times X$. We recursively define the distribution:

$$\mathcal{D} := \perp \times \texttt{enc}(\mathcal{D}_X) \quad | \quad \texttt{psh}(d \in \mathcal{D}, x \in \mathcal{D}_X) \times \texttt{enc}(x)$$

meaning $\mathcal{D}$ is a stack of arbitrary size paired with an encoded element from $\mathcal{D}_X$.

Then `psh` and `pop` have to obey the following rules:



For completeness, and to illustrate the ergonomic necessity of our diagrammatic notation, we display the formulaically obtained hyperparameterised objective function of `stack` in standard notation below. The appeal of the diagrammatic notation will become even more obvious in Section 3.

$$\text{argmin}_{\phi,\psi,\chi}\left(\underbrace{\alpha \mathbf{D}_1(\texttt{pop}_\phi(\perp_\chi), (\perp_\chi, 0))}_{\text{Empty}} + \underbrace{\beta \mathbb{E}_{(s,x)\sim\alpha}[\mathbf{D}_2(\texttt{pop}_\phi(\texttt{push}_\psi(s,x)), (s,x))]}_{\text{PshPop}} \right)$$

Where ϕ, ψ, χ are the parameters of `push`, `pop` and $\perp$ to be learnt; α, β are hyperparametric positive weighting coefficients for each task; $\mathbf{D}_1, \mathbf{D}_2$ are hyperparametric statistical divergences for $S \times X$.

Given the purely demonstrational role of this experiment, we will do a proof-of-concept on a simple analytic dataset (Figure 3) inspired by Spriteworld (Watters et al., 2019) consisting of images depicts a single shape with varying properties. We observe that the stack works exactly as expected (see Figure 2). For additional implementational details, see Section C.1.

While the basic `stack` task is not particularly interesting, it does provide a fully differentiable data structure which may be useful in composite with other tasks.

3 The `manipulation` task

In this section, we design and theoretically analyse `manipulation`, **a novel task which aims to view and edit a property of data without explicit guidance**. In Section 4, we will experimentally validate the task. There are many models that behave like `manipulation`, e.g. unsupervised sentiment translation (Li et al., 2018; Sudhakar et al., 2019) in the text domain and prompt-based photo editing (Hertz et al., 2022; Kawar et al., 2023) in the image domain. While most of these approach this question on the architectural level to impose the desired behaviour, we will instead approach this problem via the objective function - constraining the learning process. To constrain the behaviour of `manipulation`, we demonstrate another ability of our framework: importing insights from computer science more broadly via the process-theoretic perspective. For

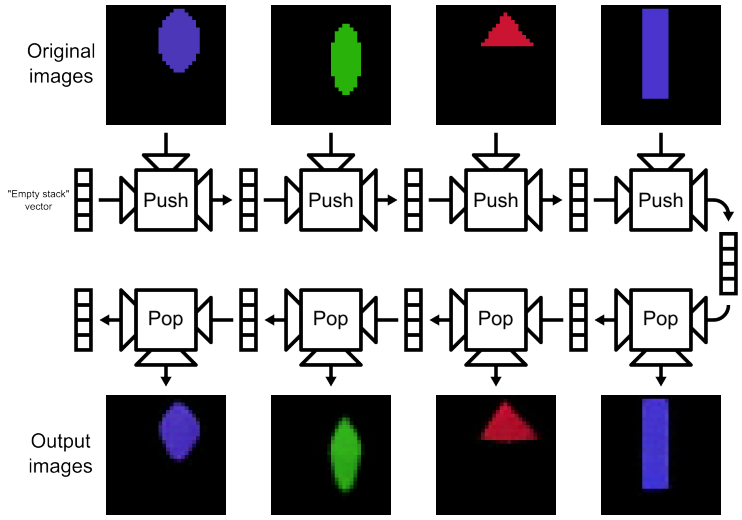
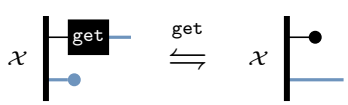


Figure 2: In this example, we train a stack (alongside an autoencoder) to store the latent vectors of Spriteworld shapes. With an image latent size 16 and stack vector size 64, it is able to retain information to faithfully restore up to 4 shapes.

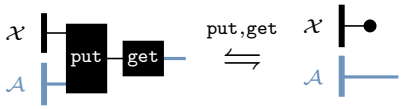
`manipulation`, we reference the field of Bidirectional Transformations, which studies consistency between different overlapping representations of data (Abou-Saleh et al., 2018). A special case is the view-update problem (Bancilhon & Spyratos, 1981) originally proposed for databases: how do we algebraically characterise reading-out and updating an attribute $a \in A$ from some data $d \in D$? There are a family of solutions called *lenses* which are parameterised by algebraic laws of varying strength (Nakano, 2021), which we take inspiration from below:

Task 3.1 (`manipulation`)**.** Let $\mathcal{D}$ be a distribution over pairs (x, a) with $x \in X$ and attribute labels $a \in A$, and let $\mathcal{D}_X$ denote its marginal on X. A modeller also specifies a target distribution $\mathcal{A}$ over desired attributes. A `manipulation` consists of a pair of operations ($\texttt{get} : X \to A$, $\texttt{put} : X \times A \to X$) that can be understood as reading and writing respectively. The two operations must satisfy the following tasks with respect to $\mathcal{D}$ and the modeller's choice of $\mathcal{A}$:



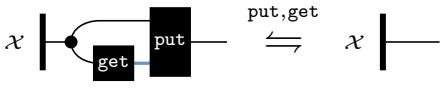
(Classify)

$$\mathbb{E}_{(d,a)\sim\mathcal{X}}[\mathbf{D}_1^A(\texttt{get}_\phi(d),\, a)]$$



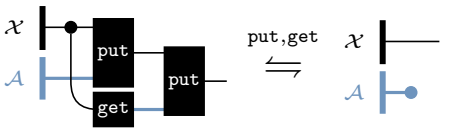
(PutGet)

$$\mathbb{E}_{(d,a')\sim\mathcal{X}|_d\times\mathcal{A}}[\mathbf{D}_2^A(\texttt{get}_\phi(\texttt{put}_\psi(d,\, a')),\, a')]$$



(GetPut)

$$\mathbb{E}_{d\sim\mathcal{X}|_d}[\mathbf{D}_1^D(\texttt{put}_\psi(d,\, \texttt{get}_\phi(d)),\, d)]$$



(Undoability)

$$\mathbb{E}_{(d,a')\sim\mathcal{X}|_d\times\mathcal{A}}[\mathbf{D}_2^D(\texttt{put}_\psi(\texttt{put}_\psi(d,\, a'),\, \texttt{get}_\phi(d)),\, d)]$$

To provide an intuition for each of the tasks, assume that the data consists of images each containing a single shape labelled with the colour of the shape. Classify allows us to use `get` to read out the colour of a shape.

PutGet says that first editing the colour of a shape (say, from red to blue) and then immediately reading out that colour returns the edited colour (blue). GetPut says that reading out the colour of a shape (say, red) followed by editing the shape to have the same colour (i.e., an edit that leaves red unchanged) is the same as doing nothing. Undoability says that edits can be undone; using the first `put` to change the colour of a shape (say from red to blue), and then editing again with a second `put` to restore the original, read-out colour of the shape (red) must restore the original image.

Once again, for completeness, we display the hyperparameterised objective function of `manipulation` in standard notation below:

$$\operatorname{argmin}_{\phi,\psi}\left(\begin{array}{c}\overbrace{\alpha\mathbb{E}_{(x,a)\sim\mathcal{D}}\left(\mathbf{D}_1^A\left(\mathtt{get}_\phi(x),a\right)\right)}^{\textsc{Classify}}+\overbrace{\beta\mathbb{E}_{(x,a')\sim\mathcal{D}_X\times\mathcal{A}}\left(\mathbf{D}_2^A\left(\mathtt{get}_\phi(\mathtt{put}_\psi(x,a')),a'\right)\right)}^{\textsc{PutGet}}\\ +\underbrace{\gamma\mathbb{E}_{x\sim\mathcal{D}_X}\left(\mathbf{D}_1^X\left(\mathtt{put}_\psi(x,\mathtt{get}_\phi(x)),x\right)\right)}_{\textsc{GetPut}}+\underbrace{\delta\mathbb{E}_{(x,a')\sim\mathcal{D}_X\times\mathcal{A}}\left(\mathbf{D}_2^X\left(\mathtt{put}_\psi(\mathtt{put}_\psi(x,a'),\mathtt{get}_\phi(x))\right)\right)}_{\textsc{Undoability}}\end{array}\right)$$

Where ϕ parameterises `get` and ψ parameterises `put`; $\alpha,\beta,\gamma,\delta$ are positive weights for each task; $\mathbf{D}_1^A,\mathbf{D}_2^A$ are divergences over attributes; $\mathbf{D}_1^X,\mathbf{D}_2^X$ act on the data space X; $\mathcal{D}_X$ is the marginal of $\mathcal{D}$ over data; and $\mathcal{A}$ is chosen so that the desired attribute values lie in $\operatorname{supp}(\mathcal{A})$. Section 4 evaluates these contracts via attribute-flip accuracy and non-target preservation checks, and **??** summarises the behavioural impact of removing individual lens laws.

3.1 Theoretical analysis

Beyond providing an intuitive language for objective functions, the proposed diagrammatic language can also be used to prove properties about a particular choice of objective function. In this section, we will prove two facts about the `manipulation` task that let us make predictions about its most likely behaviour after training.

First, we argue that an idealised model satisfying the `manipulation` task — in which attribute information is fully disentangled from the rest of the data — is inherently challenging to realise in practice. This perspective differs from global disentanglement frameworks such as β-VAE (Higgins et al., 2017), which seek latent factorisations up to dataset-wide symmetries, whereas our contracts only demand local, per-instance controllability. This is because, in complex data domains, isolating and inverting attribute-specific structure often entails recovering detailed, latent factors of variation. Such a disentangled inversion effectively requires the system to act as a conditional generator that precisely modifies only the attribute of interest while leaving all other features unchanged.

Despite this, we still expect the `manipulator` to perform well in realistic settings. Specifically, we show that the manipulation objective refines the structure of a `CycleGAN`, meaning that — under ideal training — it should match or exceed the behavioural guarantees of a standard `CycleGAN` system. Whether such performance is actually achieved in practice depends on empirical training success, which we investigate in Section 4.

3.1.1 Proof - Bayesian inversion

In the ideal case, to solve `manipulation`, a model would completely separate the information about the attribute it is manipulating from all other information. If it managed to do this, it could manipulate exactly the attribute while leaving everything else as is. For this to be possible, there must exist a complement datatype C such that our original data distribution $\mathcal{D}$ can be separated into two independent distributions $\mathcal{A}$ and $\mathcal{C}$. Assuming that this is possible, we will show that (a) being able to do such a separation would freely give a `manipulator` and (b) that this `manipulator` would refine `Bayesian inversion`, a task that is known to be difficult for complex data domains.

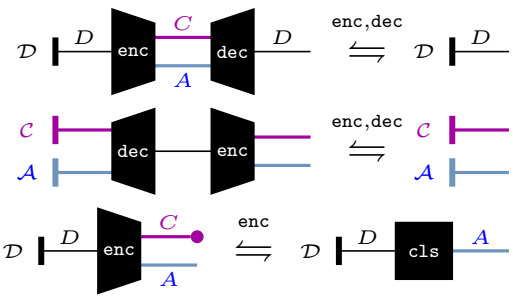
First, we define:

Definition 3.2 (Balanced entropy of an attribute)**.** Given a distribution of data $\mathcal{D}$ on space D, we say that a distribution $\mathcal{A}$ over space A represents an *entropy-balanced attribute* of $\mathcal{D}$ if there exists a complement type

C with distribution $\mathcal{C}$ such that we have the equality in distributions $\mathcal{D} = \mathcal{C} \times \mathcal{A}$ up an isomorphism of the underlying spaces $D \simeq (C \times A)$.

If there exists such an isomorphism, then we know that there must exist two functions that observe the isomorphism. We have:

Definition 3.3 (Latent split autoencoder)**.** Let the attribute A induce a balanced attribute on some distribution $\mathcal{D}$ on data D, and $\texttt{cls} : D \to A$ be a perfect classifier for the attribute A. Then the functions $\texttt{enc} : D \to C \times A$ and $\texttt{dec} : D \to C \times A$ that observe this isomorphism between the two underlying spaces satisfy the following three conditions:



(1)

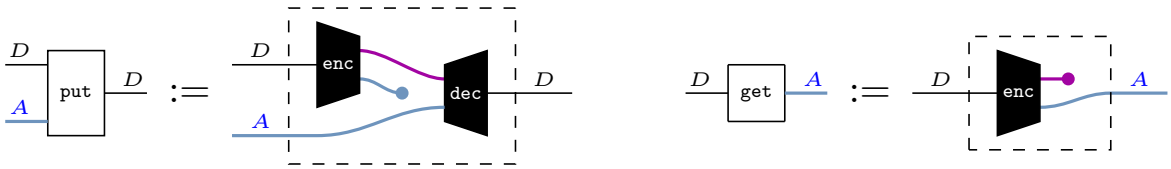
(2)

(3)

The two functions are suggestively named *enc* and *dec* as the first condition indeed specifies that they have to act as an autoencoder on $\mathcal{D}$. We will show that given such an idealised autoencoder, we can freely, i.e. without any additional training, satisfy `manipulation`:

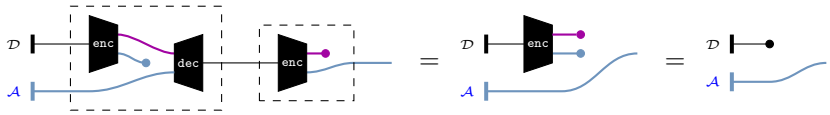
Proposition 3.4 (`Manipulation` from `latent split autoencoder`)**.** Let $\mathcal{D}$ be a distribution with a balanced attribute A. Then a `latent split autoencoder` refines `manipulation`.

Proof. We can define:



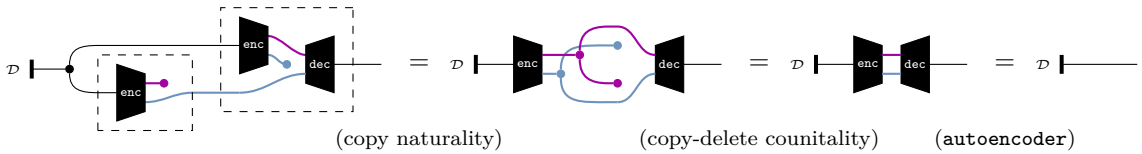
We now have to show that this indeed satisfies `manipulation`.

For this, we show that the `put` and `get` observe the restrictions of `manipulation`:
PutGet:



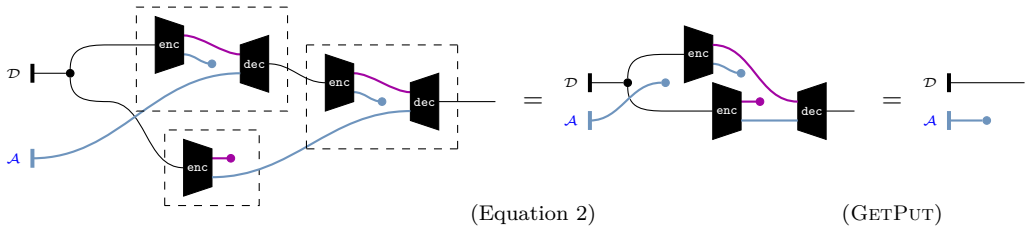
(Equation 2) (Delete naturality)

GetPut:



(copy naturality) (copy-delete counitality) (`autoencoder`)

UNDOABILITY:



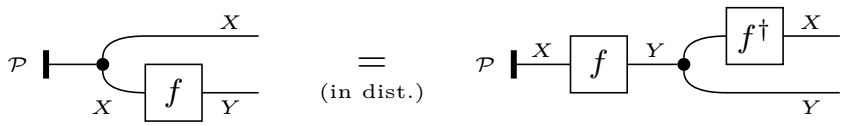
□

Thus, solving the information separation problem would also give a solution to `manipulation`.

We will now show that this is difficult by proving that this `manipulation` freely provides a Bayesian inversion to a `classifier` from D to A.

We define:

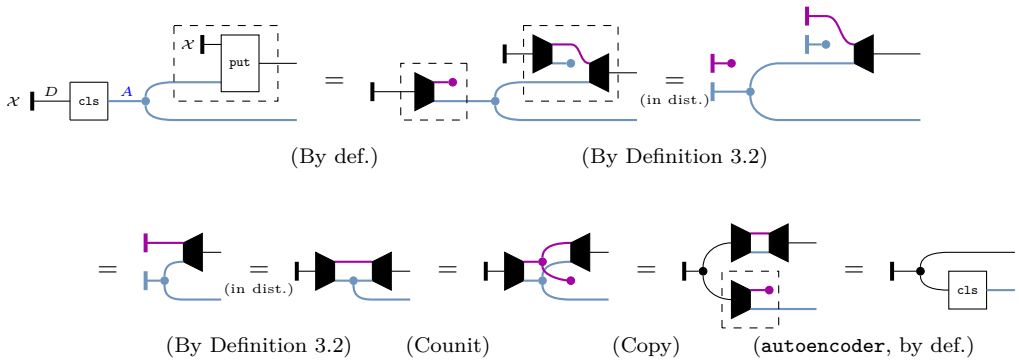
Definition 3.5 (Bayesian Inversion in Markov Categories)**.** The Bayesian inversion (Cho & Jacobs, 2019; nLab authors, 2024a) of a stochastic map $f : X \to Y$ with respect to a distribution $\mathcal{P}$ on X is a stochastic map $f^\dagger : Y \to X$ such that, in distribution:



We can show:

Proposition 3.6 (`Manipulation` as Bayesian inversion)**.** If a deterministic discriminative classifier `cls` : $D \to A$ induces a balanced attribute `cls`$(\mathcal{D})$ over A with respect to D, then the `manipulator` specified above refines Bayesian inversion `cls`$^\dagger : A \to D$.

Proof. We show that the `put` composed with an independent copy of the data source $\mathcal{D}$ is the Bayesian inversion of `cls`.



□

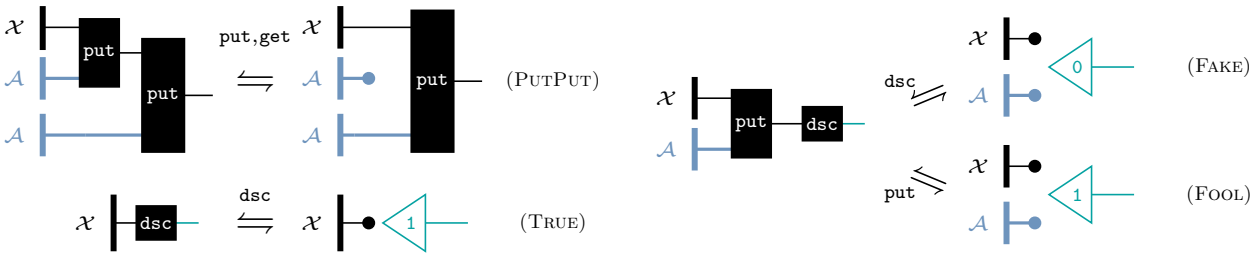
As such, we have shown that the idealised solution to the `manipulation` problem would also give a solution to Bayesian inversion, which is generally considered a challenging problem in machine learning. Given this inherent difficulty, one might reasonably question why we expect this setup to succeed. Before providing empirical evidence, we will first develop theoretical intuition by relating `manipulation` to the established task of `CycleGAN`.

3.1.2 Proof - CycleGAN

`CycleGAN`s solve a similar task as `manipulation`, translating between two distributions. In fact, with the additional regularisation terms, `strong manipulation` is a refinement of `CycleGAN`, giving us more guarantees by avoiding certain failure cases.

To prove this relationship formally, we have to add additional regularisation terms to the `manipulation` task. While these are theoretically justified, in practice they are less train stable. Therefore, in our experiments, we will use the vanilla `manipulation` as a starting point.

Task 3.7 (`strong manipulation` add-ons)**.** The `strong manipulation` task consists of all the tasks contained in `manipulation`, we well as:



The PutPut task (which, paired with PutGet and GetPut is strictly stronger than Undoability in that it is algebraically implied) says that the effect of `put`ting twice is the same as discarding the effect of the first edit and only keeping the last edit. In conjunction with PutGet and GetPut, this creates what is known in the literature as a *very well-behaved lens*. The True, Fake, and Fool tasks introduce a discriminator component `dsc`, which forms a `GAN` pattern with respect to `put` as the generator. When well-trained, this forces the outputs of `put` to lie in-distribution. As in general there are no algebraic or equational laws that characterise arbitrary distributions of data, using `GAN`s in this way is a generic recipe for shaping outputs of generators to behave well in-distribution.

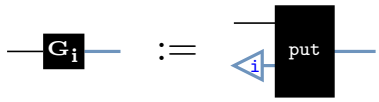
To show that `strong manipulation` refines `CycleGAN`, we require one additional assumption: a generator in a generative-adversarial setting is optimal if and only if its output distribution is equal to the original distribution. This is a natural assumption for such settings, and has been proven formally (Goodfellow et al., 2020, Theorem 1) by Goodfellow et al for an appropriate choice of measure of statistical divergence. We may now prove a weaker variant of refinement, relative to optimal (rather than perfect) tasks.

Proposition 3.8 (Strong Manipulation and CycleGAN)**.** Given appropriate choices of statistical divergence measures, optimal solutions to the strong manipulation task can be used to construct optimal solutions to the CycleGAN task.

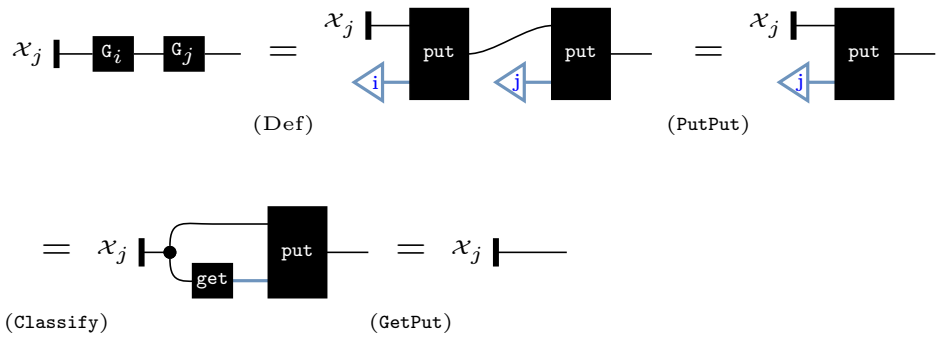
Proof. Let $\mathcal{D}_0, \mathcal{D}_1$ be two distributions over the same type. We create a combined space $\mathcal{D} = \{(x, i) \mid x \in \mathcal{D}_i, i \in \{0, 1\}\}$ where the label i indicates which distribution the data point came from.

Assume we have functions (`get`, `put`, `dsc`) that optimally satisfy the strong manipulation task for this new dataset. Then we show that we can construct an optimal `CycleGAN` from these functions.

For $i \in \{0, 1\}$, $j = 1 - i$, we can construct generators G_0 and G_1 as follows:

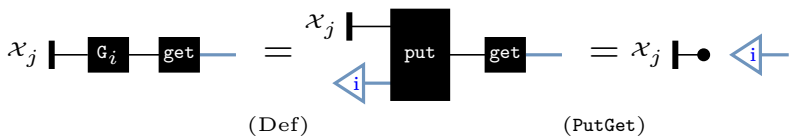


First, we show that the cycle-consistency requirements of CycleGAN are satisfied:



Next, we show that these generators are optimal in the GAN sense. As outlined above, we say a generator is optimal if and only if its output distribution matches the target distribution. Since `put` is optimally trained with the discriminator `dsc`, $\texttt{put}(x, i)$ produces outputs indistinguishable from $\mathcal{D}_i$.

By the PutGet property:



Combined with the Classify property, this ensures that G_i produces outputs that belong to the correct distribution $\mathcal{D}_i$.

Thus, the generators G_0 and G_1 derived from strong manipulation are optimal solutions to the CycleGAN task.

□

However, the converse does not hold: CycleGAN is not a refinement of strong manipulation. For example, a CycleGAN could learn generators that apply a consistent reversible transformation (like horizontal flipping) while preserving cycle-consistency, but such a solution would violate the PutPut property of strong manipulation. This demonstrates that strong manipulation provides stronger guarantees about the behavior of its learned functions than CycleGAN does.

Despite the additional guarantees, `strong manipulation` does not guarantee that it indeed only changes as little as necessary to go from one distribution to the other. To showcase this, we can consider a degenerate example where no unique solution to `manipulation` can exist: imagine data with four objects—circle, square, red and green. There is no unique mapping between shapes and colors. Without further information, it is therefore impossible to say what it would mean to change as little as possible to go from shapes to colors. Yet, despite these theoretical concerns, in practice, `manipulation` often converges to the desired behavior, similar to `CycleGAN`, which has even fewer guarantees.

As a comparison to a broader vein in the literature than CycleGAN, we can consider Disentangled Representation Learning[3] (DRL). Following Wang et al. (2024), two common formulations of DRL are via Bengio et al. (2013) (the "intuitive" definition), and Higgins et al. (2018) (the group-theoretic definition). The latent-split autoencoder construction we have presented is a flat, vector-wise DRL formalisation of the first definition, disentangling a single attribute. Proposition 3.4 constructs a put as a $m(x, y) = y$ semigroup-action on the disentangled attribute space, which conforms with Higgins et al's group-action definition as a conservative generalisation. A converse refinement (and hence, task equivalence) holds if we are granted a style-projection map $\pi : D \to D/\sim$, where $D/\sim$ is the input space D quotiented by the equivalence relation $d \sim d' \triangleq \forall a_{\in A} : \texttt{put}(d, a) = \texttt{put}(d', a)$. As a sketch, this obtains $D \simeq D/\sim \times A$ by the correspondences $\texttt{enc}(d) \mapsto (\pi(d), \texttt{get}(d))$ and $\texttt{dec}(s, a) \mapsto \texttt{put}(s, a)$.

[3]We thank an anonymous reviewer for bringing this field to our attention for a constructive comparison.

While a comprehensive comparative study is beyond the scope of this work, we remark in passing that Manipulation appears to be different in aims from DRL, and this difference in intent manifests downstream as differences in use. In essence, DRL seeks global coordinate charts of data with low interdependence and invariances, whereas Manipulation aims merely at local, per-instance on-manifold control. Intuitively, the former is asking for more than the latter, since it is possible to navigate terrain locally without a global map, and that is what Proposition 3.4 demonstrates formally. In practice, the unsupervised factor discovery of DRL aims to obtain block-diagonally separate latents identifiable up to a gauge choice of rotations, whereas Manipulation anchors the attribute coordinate and allows for detection and enforcement of cross-factor causal influence by edit-commutation, reminiscent of Scholkopf et al. (2021)'s causal DRL.

4 Experimental validation of `manipulation`

In this section, we provide simple but illustrative demonstrations of the `manipulation task`, designed primarily to validate theoretical predictions about task-driven behaviour in controlled settings. While these examples do not represent state-of-the-art performance, they clearly illustrate how our framework systematically predicts model behaviour from task structure alone.

We will first provide some toy examples of the vanilla `manipulation` task, a setting in which we can also adapt to distributions without balanced entropy. Then we will showcase the `manipulation` task with more realistic data, manipulating attributes of faces. For technical implementation details, see Appendix C.

4.1 Experimental Results I: Simple attributes of synthetic and real-world data

We first demonstrate initial proofs-of-concept of the `manipulation` task on a simple analytic dataset (Figure 3) inspired by Spriteworld (Watters et al., 2019), and on MNIST (Figure 4). In the former, each image depicts a single shape with varying properties, and is labelled by two attributes: shape – circle, square or triangle – and colour – red, green or blue. For each attribute, we train a `get/put` pair according to the `manipulation` task specification.

These initial demonstrations on synthetic data illustrate the potential generalisation properties theoretically predicted by our formal task analysis, particularly the ability of models to interpolate smoothly between attribute states. We emphasise that our goal here is not performance optimisation, but rather to transparently illustrate theoretical predictions in a comprehensible setting.

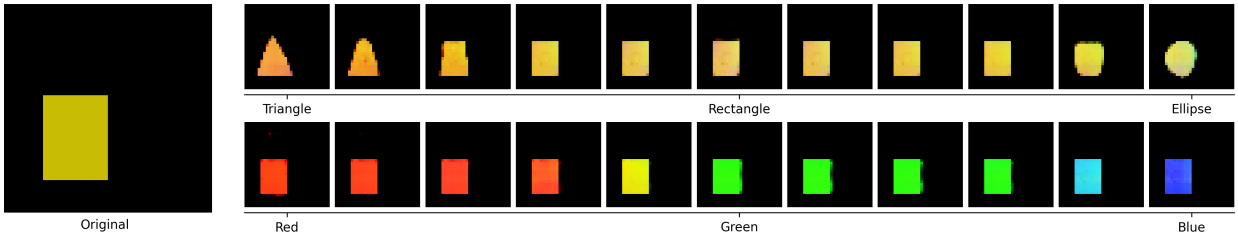


Figure 3: An input Spriteworld image alongside a spectrum of outputs exhibiting the ability of the `put` to manipulate a single attribute of the input while preserving its other properties. Additionally, the model is able to generalise by interpolating to attribute values unseen during training, in this case producing orange and cyan shapes, whereas during training, it only sees red, green or blue shapes. (further details in Section C.2)

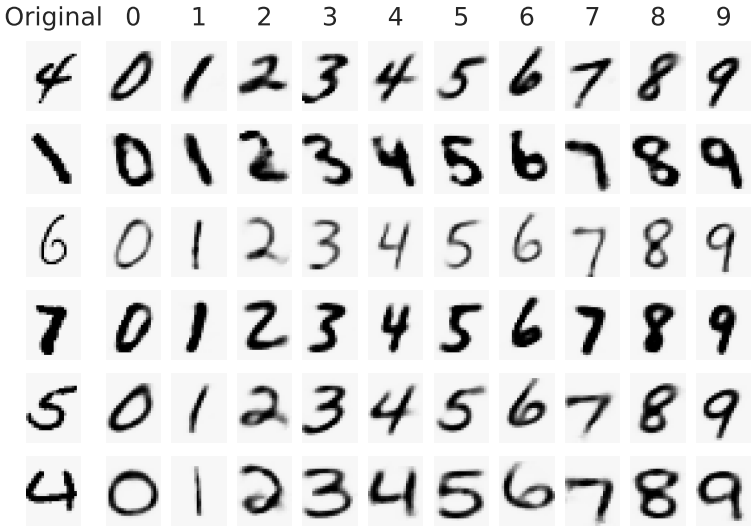


Figure 4: Outputs of a `put` trained against an MNIST classifier. The `put` preserves several graphological aspects, such as stroke weight, slant, and angularity. This represents qualitative evidence to support our prediction that `put` as a class-conditioned generative model behaves as a style-preserving edit.

4.2 Experimental Results II: Derived attributes of synthetic data

The following demonstrations explore theoretically anticipated challenges in manipulating complex, derived attributes—particularly entropy imbalances known to cause issues in more practical, large-scale contexts. We intentionally simplify and explicitly control our synthetic scenario (Spriteworld) to provide clear, instructive validation of our theoretical insights into the interplay between task constraints and model behaviours.

Often in practice we are interested in complex, non-explicit attributes that are derived from those labelled in the data: for example, "eligibility for a loan" may be derived from other explicit attributes of people in a database by an operationally opaque classifier, with unknown range, distribution, and dependencies on other attributes. A known challenge in manipulating derived attributes is unequal entropy in attribute classes (Chu et al., 2017), which may cause models such as CycleGANs to hide data imperceptibly, making them particularly vulnerable to adversarial attacks. Various solutions have been proposed, including masks (Wu et al., 2024), blurring (Fu et al., 2019) and compression (Dziugaite et al., 2016). We demonstrate via a modification of `manipulation` (Task 3.1,Figure 6) that our framework permits the design and implementation of end-to-end approaches to editing complex attributes without interventions on the data.

$$bc = \begin{cases} \min(1, cs + 0.6) & \text{if } \textit{shape} = \textit{circle} \\ \min(0.8, \max(0.2, cs)) & \text{if } \textit{shape} = \textit{square} \\ \max(0, cs - 0.6) & \text{if } \textit{shape} = \textit{triangle} \end{cases}$$

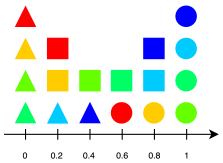
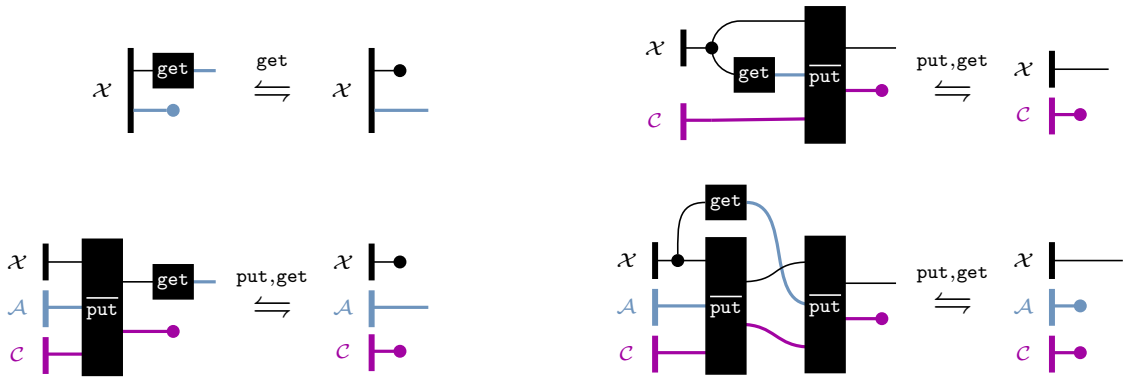


Figure 5: To illustrate the concepts of derived attributes and unequal entropy, consider an attribute on the Spriteworld data called *blue-circleness*, which broadly measures how similar a shape is to a blue circle. We define *blue-circleness* (bc) as a function of explicit attributes *shape* and *colour*; we assign a continuous colour score $cs \in [0, 1]$ based on the hue, where red $= 0$ and blue $= 1$. To illustrate unequal entropy in this example, the class 0 has higher entropy than 0.4 because there are more shapes that have bc-value 0. So manipulating a shape with bc-value 0 to 0.4 must lose information.

Task 4.1 (`complement manipulation`)**.** Inspired by the complement of symmetric lenses (Hofmann et al., 2011), we introduce a complement C to `put`, changing its type to $S \times L \times C \to S \times C$. Let $(d, a) \sim \mathcal{X}$ be a distribution over some data $d \in D$, each labelled with an attribute $a \in A$. `complement manipulation` consists of a pair ($\texttt{get} : D \to A$, $\overline{\texttt{put}} : D \times A \times C \to D \times C$) fulfilling the following rules:



The idea of the complement is that it provides the manipulation with a scratchpad C to keep track of additional data. As none of the tasks check the output of the complement, the `put` and `get` can use it freely to store relevant data.

While the complement manipulator successfully demonstrates our theoretical predictions at small scales, we stress that this controlled scenario is illustrative of underlying theoretical mechanisms rather than directly

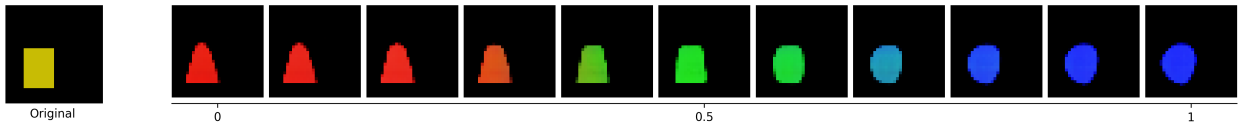


Figure 6: A `complement manipulator` (Task 4.1) can manipulate derived attributes such as *blue-circleness*, by using the complement as a scratchpad to record a correspondence between data points (further details in Section C.2) while preserving attributes such as position and size.

indicative of performance at scale. The value here lies primarily in transparently validating predictions about how task structure can address known issues (e.g., entropy imbalance) without requiring explicit architectural interventions.

4.3 Experimental results III: Interpretability applications on real-world data

In this section, we further illustrate theoretical predictions of our task framework by applying manipulation tasks to a somewhat more complex, yet still controlled, real-world dataset (CelebFaces Attributes). These experiments demonstrate how specific theoretical insights, such as the introduction of linearity constraints, can systematically yield interpretable and theoretically anticipated behaviours in practice, albeit at exploratory scales.

In the same way that we would expect a latent space "filtered through" the probabilistic structure of Gaussians to yield good sampling properties (Example 2.5), we would expect that enforcing linear structure on the latent space would yield linear properties. So we specialised the `put` to be a simple vector addition in the latent space of an autoencoder (Figure 7), for the relatively complex *Large-scale CelebFaces Attributes* dataset. We found that restricting `put` to be linear in this way not only increased training stability, but indeed exhibited continuous interpolation in generated outputs between normally discrete class labels (Figure 8), and class-sensitive separation of latent space embeddings in the autoencoder. We consider this to be compelling evidence that since our framework is agnostic, implementation details may be engineered to obtain additional desirable properties without compromising behavioural specifications.

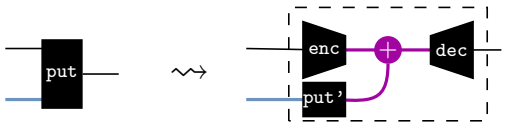


Figure 7: Recalling that architectural choices are a form of specialisation by diagrammatic substitution, the linear `put` is a specialisation of a generic `put` as an `autoencoder` task-bound pair of learners `enc` and `dec`, along with a `put'` that computes single shift vector to be added into the latent space, depending only on the label value. `enc`, `dec`, and `put'` are trained simultaneously along with the `manipulation` tasks, and intuitively this pressures the autoencoder pair to adapt their latent representations to fit the needs of the broader `manipulation` task.

These illustrative results support our theoretical claim that structured tasks systematically induce predictable model behaviours: in this case, linearity enabling continuous latent interpolations. Although the complexity of the chosen dataset moves beyond simple synthetic data, we explicitly note these demonstrations remain exploratory validations intended primarily as proofs-of-principle for theoretical predictions rather than definitive large-scale benchmarks.

Taken together, the illustrative examples presented across synthetic and small-scale real-world datasets serve primarily as concrete demonstrations of our framework's theoretical predictive power, rather than comprehensive empirical evaluations of practical performance or scalability. Such demonstrations confirm the

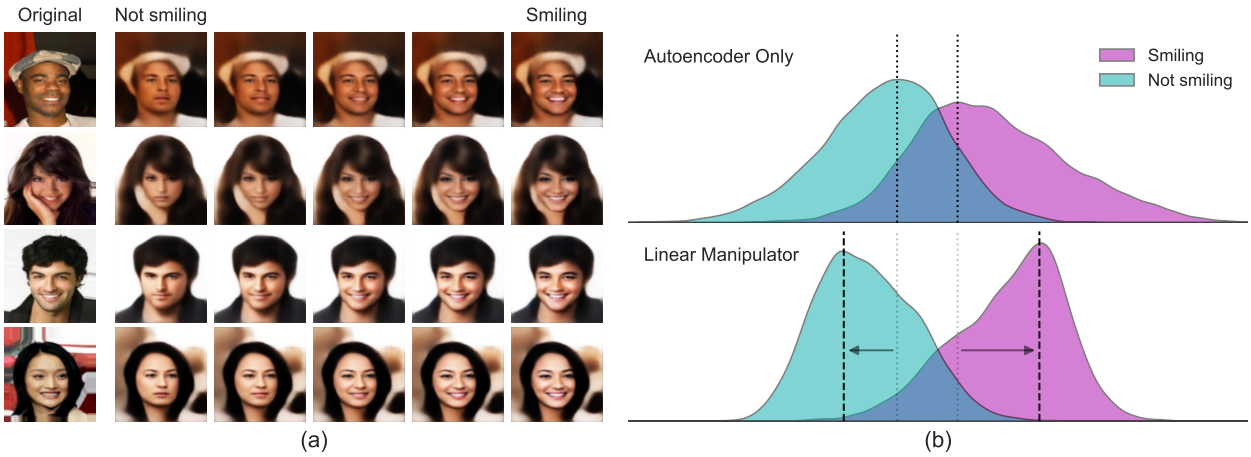


Figure 8: *Left:* Outputs of a linear `manipulator` trained on face image data paired with a binary smile/no-smile label. The remarkable aspect of this experiment is that the original data only carried binary smile/no-smile labels, and that the linear structure in the specialisation of the `put` admits continuous interpolation. *Right:* A comparison of the spread of the latent embeddings of images from the validation set when pre-training an autoencoder and then training a (linear) classifier on the latent space (top), vs. when trained with a linear `manipulator` (bottom). We find that linear put automatically separates latent space embeddings of classes: the graphs depict the relative density of embeddings along the direction of the classifiers' weight vectors, normalised so that each combined data spread is centred and has unit variance (details in Section C.3).

clarity and utility of our theoretical framework, and explicitly highlight opportunities for future exploration at larger scales and across broader practical domains.

5 Concluding discussion

5.1 Summary

We introduced a diagrammatic language for representing and reasoning about the behaviour of machine learning models in terms of tasks, viewed as the essential data of objective functions. By leveraging category theory and string diagrams, our work establishes a cross-disciplinary formal bridge between theoretical computer science and practical machine learning, providing new conceptual tools for analyzing ML systems and permitting the transfer of insights between traditionally separate fields.

The proposed framework allows capturing existing tasks in machine learning, providing intuitive insights rooted in mathematical rigour. We identify a set of widespread and well-understood tasks, which we call patterns. We can analyse some tasks as composites of patterns (Example 2.12) while other tasks can be understood as specialisations of patterns (Example 2.5). The rewrite system inherent to string diagrams, allows us to identify relationships between different tasks and formalise intuitions (Proposition 2.17, Proposition 3.8).

Beyond theoretical insights, the proposed language also allows the creation of new training paradigms. As preliminary empirical validation of our theory's utility and potential, we introduced a novel task type called `manipulation` that produces a class-conditioned and style-preserving generative model counterpart for a given classifier. In the image domain, we were able to verify predicted behaviours (Section 4.1), and we demonstrated the ability to design novel end-to-end capabilities, such as end-to-end editing of complex attributes (Section 4.2) and the imposition of linear structure on latent space representations (Section 4.3), which allowed continuous interpolation between discrete class labels on real data, and separated latent space embeddings of different classes. Notably, this was achieved without adversarial training conditions, random sampling, preprocessing of data, or hardcoded interventions in the architectures, i.e.: *our framework enables capable, small-scale, and training-stable models.*

5.2 Relation to extant work

Regarding our nascent theoretical framework as a whole, the style of engineering beginning from tasks is already common practice in many fields of ML, and we sought here to place these practices on a more rigorous footing, and to probe their strengths and limitations. Our mathematical approach draws broadly from the field of Applied Category Theory (Fong & Spivak, 2019a), particularly in the use of string diagrams for the higher-algebraic data of concurrently and serially composed functions, which enables compact representation and reasoning with otherwise cumbersome symbolic equivalents when dealing with multiple learners in tandem. To our knowledge, our concern with the composition of tasks among many learners distinguishes our aims and formal choices from approaches that employ similar mathematical formulations, both within the category-theoretic literature (cf. Gavranović et al. (2024)), and without (cf. the variational generalisation of Bayesian inference presented in Knoblauch et al. (2022)).

While our approach is essentially neurosymbolic in spirit, it does not fit neatly into the mainstream triad of neurosymbolic approaches (d'Avila Garcez & Lamb, 2020); we do not encode symbolic data for neural operations, nor do we interface neural approaches with symbolic engines, nor are we hardcoding expert knowledge representations. Moreover, our aims differ: while neurosymbolic approaches often seek to manipulate symbolic data systematically by neural means, our framework operates at a higher level of abstraction, seeking to use the systematicity of higher-algebraic equational characterisations as a means to shape the neural ends. Hence our perspective may complement existing approaches to structure in machine learning.

Regarding `manipulation` in particular, this was to our knowledge the first practically demonstrated synthesis of insights from Bidirectional Transformations as a subfield of database theory (Abou-Saleh et al., 2018) with ML. While explicitly neurosymbolic approaches have been tried for similar editing tasks before (see e.g. Smet et al. (2023)), owing to the influence of database theory in our approach, to our knowledge our statement and execution of this task enjoys the maximal permissible generality and implementation agnosticism among similar attribute-editing tasks, without sacrificing rigour and systematicity.

5.3 Limitations and Prospects

5.3.1 Scaling and Empirical Validation

Our experimental results are necessarily preliminary and limited in scale, intended primarily as sanity checks and first demonstrations rather than definitive benchmarks. Recognising this, we note explicitly that further empirical exploration, while beyond our current scope, would be valuable and necessary to fully assess practical scalability and broader utility. Future work should rigorously test our theoretical predictions at larger scales and in comparison with established state-of-the-art frameworks.

Concerning `manipulation` in particular, an immediately evident limitation of this practical demonstration is a lack of exploration of how the difficulty of training such ensembles of learners behaves at scale, with respect to more complex and multimodal datasets, and with a wider range of architectures. While none of the products of our experimentation are state-of-the-art with respect to specific applications, we believe the variety and promise of these results serve as a compelling validation of our theoretical framework's utility and potential.

Concerning applications of `manipulation` beyond the image domain, we report on some sketch experiments in sentiment-manipulation on text in Appendix D, where we also comment on the nature of technical difficulties to be overcome in the application of `manipulation` to complex domain data, and offer an explanation for mode collapses observed during training by empirically relating `manipulation` to other generative classification approaches.

5.3.2 Complexity and Theoretical Formalism

We acknowledge that our reliance on categorical formalisms and abstract mathematical structures, while foundationally valuable, may impose an initial barrier to practitioners less familiar with these tools. However, we believe that such rigour is precisely what enables clear theoretical insights and structured reasoning about model behaviour. To aid accessibility, we have provided intuitive explanations accompanying formal constructs and a friendly diagrammatic syntax that can be used productively without a detailed understanding of the formal semantics. We anticipate practitioners can benefit from engaging with these theoretical underpinnings incrementally.

Addressing the theoretical framework of tasks more broadly, our reliance on equational characterisations is double-edged. On one hand, it is uncommon to find such characterisations of mathematical systems of interest as they are usually defined by more direct means, and this presents a theoretical limitation. On the other hand, it appears that the strength of equational characterisations when applied to ML lies in imposing structure on "the way to learn to solve a problem" rather than on the solutions or problems themselves (Sutton, 2019). This suggests promising future possibilities of our mathematical framework in bridging structural-symbolic approaches from computer science more broadly with methods that can effectively leverage computation.

5.3.3 Theoretical Gaps

While we have demonstrated that, in certain cases, tasks can determine behaviours, there is a theoretical gap in the converse analysis of behaviours of trained models in terms of their basic tasks.

Our theoretical framework currently best handles static, equilibrium conditions of task-optimisation, providing robust insights into structural behaviour under idealised limits. Dynamic adversarial interactions, such as those arising in GAN-based architectures, introduce complexity that our present theoretical treatment explicitly acknowledges but does not fully resolve. This represents a fruitful avenue for future theoretical expansion, underscoring the potential for this framework to evolve and integrate more sophisticated dynamical systems analysis.

Future theoretical developments will also seek to incorporate other aspects of ML: for example, relating to work that focuses on the choice of model architecture (Khatri et al., 2024) and interactions with the underlying data distribution (Bronstein et al., 2021). While our current experiments focus on demonstrating our framework's validity, future practical developments will explore applications to more complex, real-world ML challenges, where we envision our approach informing areas such as AutoML, interpretable AI, and formal verification of ML systems: the compositional nature of our task-based framework naturally aligns with neural architecture search by potentially informing principled search strategies for optimal model architectures, and the explicit representation of model behaviours as equational constraints could enhance interpretability and facilitate formal verification.

References

Faris Abou-Saleh, James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. *Introduction to Bidirectional Transformations*, pp. 1–28. Springer International Publishing, Cham, 2018. ISBN 978-3-319-79108-1. doi: 10.1007/978-3-319-79108-1_1. URL https://doi.org/10.1007/978-3-319-79108-1_1.

Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language: Towns, buildings, construction*, volume 2 of *Center for Environmental Structure Series*. Oxford University Press, New York, 1977.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic Variational Video Prediction, March 2018. URL https://doi.org/10.48550/arXiv.1710.11252.

John C. Baez and Jade Master. Open Petri Nets. *Mathematical Structures in Computer Science*, 30 (3):314–341, March 2020. ISSN 0960-1295, 1469-8072. doi: 10.1017/S0960129520000043. URL http://arxiv.org/abs/1808.05415.

Francois Bancilhon and Nicolas Spyratos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, dec 1981. ISSN 0362-5915. doi: 10.1145/319628.319634. URL https://doi.org/10.1145/319628.319634.

Kent Beck and Ward Cunningham. Using Pattern Languages for Object-Oriented Programs. Technical Report CR-87-43, September 1987.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/e995f98d56967d946471af29d7bf99f1-Paper.pdf.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, August 2013. ISSN 1939-3539. doi: 10.1109/TPAMI.2013.50.

Guillaume Boisseau and Paweł Sobociński. String Diagrammatic Electrical Circuit Theory. *Electronic Proceedings in Theoretical Computer Science*, 372:178–191, November 2022. ISSN 2075-2180. doi: 10.4204/EPTCS.372.13. URL http://arxiv.org/abs/2106.07763.

Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. A Categorical Semantics of Signal Flow Graphs. In *CONCUR 2014 - Concurrency Theory - 25th International Conference*, Rome, Italy, September 2014. URL https://hal.science/hal-02134182.

Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Interacting Hopf Algebras. *Journal of Pure and Applied Algebra*, 221(1):144–184, January 2017. ISSN 00224049. doi: 10.1016/j.jpaa.2016.06.002. URL http://arxiv.org/abs/1403.7048.

Filippo Bonchi, Robin Piedeleu, Pawel Sobociński, and Fabio Zanasi. Graphical Affine Algebra. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–12, June 2019. doi: 10.1109/LICS.2019.8785877. URL https://doi.org/10.1109/LICS.2019.8785877.

Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7327–7347, November 2022. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2021.3116668. URL https://doi.org/10.48550/arXiv.2103.04922.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating Sentences from a Continuous Space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 10–21, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1002. URL https://doi.org/10.48550/arXiv.1511.06349.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. URL https://doi.org/10.48550/arXiv.2104.13478.

Kenta Cho and Bart Jacobs. Disintegration and Bayesian Inversion via String Diagrams. *Mathematical Structures in Computer Science*, 29(7):938–971, August 2019. ISSN 0960-1295, 1469-8072. doi: 10.1017/S0960129518000488. URL https://doi.org/10.48550/arXiv.1709.00322.

Casey Chu, Andrey Zhmoginov, and Mark Sandler. Cyclegan, a master of steganography. *arXiv preprint arXiv:1712.02950*, 2017. URL https://doi.org/10.48550/arXiv.1712.02950.

Lorenzo Ciampiconi, Adam Elwood, Marco Leonardi, Ashraf Mohamed, and Alessandro Rozza. A survey and taxonomy of loss functions in machine learning, January 2023. URL https://doi.org/10.48550/arXiv.2301.05579.

Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, April 2011. doi: 10.1088/1367-2630/13/4/043016. URL https://dx.doi.org/10.1088/1367-2630/13/4/043016.

Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, Cambridge, 2017. ISBN 978-1-107-10422-8. doi: 10.1017/9781316219317. URL https://www.cambridge.org/core/books/picturing-quantum-processes/1119568B3101F3A685BE832FEEC53E52.

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical Foundations for a Compositional Distributional Model of Meaning, March 2010. URL http://arxiv.org/abs/1003.4394.

Geoffrey SH Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. In *Programming Languages and Systems: 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings*, pp. 1–28. Springer International Publishing Cham, 2022. URL https://doi.org/10.48550/arXiv.2103.01931.

Artur d'Avila Garcez and Luis C. Lamb. Neurosymbolic AI: The 3rd Wave. https://arxiv.org/abs/2012.05876v2, December 2020.

Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M. Roy. A study of the effect of JPG compression on adversarial images, August 2016. URL https://doi.org/10.48550/arXiv.1608.00853.

Brendan Fong and David I. Spivak. *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press, 1 edition, July 2019a. ISBN 978-1-108-66880-4 978-1-108-48229-5 978-1-108-71182-1. doi: 10.1017/9781108668804.

Brendan Fong and David I. Spivak. Supplying bells and whistles in symmetric monoidal categories., August 2019b. URL https://doi.org/10.48550/arXiv.1908.02633.

Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, January 1976a. ISSN 0092-7872. doi: 10.1080/00927877608822127.

Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, January 1976b. ISSN 0092-7872, 1532-4125. doi: 10.1080/00927877608822127. URL https://doi.org/10.1080/00927877608822127.

Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, August 2020. ISSN 00018708. doi: 10.1016/j.aim.2020.107239. URL https://doi.org/10.1016/j.aim.2020.107239.

Tobias Fritz, Tomáš Gonda, and Paolo Perrone. De Finetti's Theorem in Categorical Probability. *Journal of Stochastic Analysis*, 2(4), November 2021. ISSN 2689-6931. doi: 10.31390/josa.2.4.06. URL http://arxiv.org/abs/2105.02639.

Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, Kun Zhang, and Dacheng Tao. Geometry-consistent generative adversarial networks for one-sided unsupervised domain mapping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Bruno Gavranović, Paul Lessard, Andrew Dudzik, Tamara von Glehn, João G. M. Araújo, and Petar Veličković. Categorical deep learning: An algebraic theory of architectures, February 2024. URL https://doi.org/10.48550/arXiv.2402.15332.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11): 139–144, 2020. URL https://doi.org/10.1145/3422622.

Nathan Haydon and Paweł Sobociński. Compositional Diagrammatic First-Order Logic. In Ahti-Veikko Pietarinen, Peter Chapman, Leonie Bosveld-de Smet, Valeria Giardino, James Corter, and Sven Linker (eds.), *Diagrammatic Representation and Inference*, Lecture Notes in Computer Science, pp. 402–418, Cham, 2020. Springer International Publishing. ISBN 978-3-030-54249-8. doi: 10.1007/978-3-030-54249-8_32. URL https://doi.org/10.1007/978-3-030-54249-8_32.

Jules Hedges. String diagrams for game theory, March 2015. URL http://arxiv.org/abs/1503.06072.

Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-or. Prompt-to-prompt image editing with cross-attention control. In *The Eleventh International Conference on Learning Representations*, 2022. URL https://doi.org/10.48550/arXiv.2208.01626.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β-vae: Learning basic visual concepts with a constrained variational framework. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a Definition of Disentangled Representations, December 2018.

Martin Hofmann, Benjamin Pierce, and Daniel Wagner. Symmetric lenses. *ACM SIGPLAN Notices*, 46(1): 371–384, 2011.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal inference by string diagram surgery. In *Foundations of Software Science and Computation Structures: 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings 22*, pp. 313–329. Springer, 2019. URL https://doi.org/10.1007/978-3-030-17127-8_18.

Bahjat Kawar, Shiran Zada, Oran Lang, Omer Tov, Huiwen Chang, Tali Dekel, Inbar Mosseri, and Michal Irani. Imagic: Text-based real image editing with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6007–6017, June 2023.

Nikhil Khatri, Tuomas Laakkonen, Jonathon Liu, and Vincent Wang-Maścianica. On the anatomy of attention, 2024.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes, December 2022. URL https://doi.org/10.48550/arXiv.1312.6114.

Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. An Optimization-centric View on Bayes' Rule: Reviewing and Generalizing Variational Inference. *Journal of Machine Learning Research*, 23:1–109, 2022.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, MA, 2009.

Juncen Li, Robin Jia, He He, and Percy Liang. Delete, retrieve, generate: a simple approach to sentiment and style transfer. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1865–1874, 2018. URL https://doi.org/10.18653/v1/N18-1169.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Robin Lorenz and Sean Tull. Causal models in string diagrams, April 2023. URL https://doi.org/10.48550/arXiv.2304.07638.

Martin Markl, Steve Shnider, and Jim Stasheff. *Operads in Algebra, Topology and Physics*. American Mathematical Society, Providence, RI, July 2007. ISBN 978-0-8218-4362-8.

Samuel G. Müller and Frank Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation, 2021.

Keisuke Nakano. A Tangled Web of 12 Lens Laws. In *Reversible Computation: 13th International Conference, RC 2021, Virtual Event, July 7–8, 2021, Proceedings*, pp. 185–203, Berlin, Heidelberg, July 2021. Springer-Verlag. ISBN 978-3-030-79836-9. doi: 10.1007/978-3-030-79837-6_11.

Andrew Ng and Michael Jordan. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

nLab authors. Bayesian inversion. `https://ncatlab.org/nlab/show/Bayesian+inversion`, June 2024a. Revision 9.

nLab authors. Markov category. `https://ncatlab.org/nlab/show/Markov+category`, July 2024b. Revision 56.

Prakash Panangaden. The Category of Markov Kernels. *Electr. Notes Theor. Comput. Sci.*, 22:171–187, December 1999. doi: 10.1016/S1571-0661(05)80602-4. URL `https://doi.org/10.1016/S1571-0661(05)80602-4`.

Dusko Pavlovic. Monoidal computer i: Basic computability by string diagrams. *Information and Computation*, 226:94–116, 2013. ISSN 0890-5401. doi: https://doi.org/10.1016/j.ic.2013.03.007. URL `https://www.sciencedirect.com/science/article/pii/S0890540113000254`. Special Issue: Information Security as a Resource.

Dusko Pavlovic. *Programs as Diagrams: From Categorical Computability to Computable Categories*. Springer, 1st ed. 2023 edition edition, September 2023. ISBN 978-3-031-34826-6. URL `https://doi.org/10.48550/arXiv.2208.03817`.

Boldizsár Poór, Quanlong Wang, Razin A. Shaikh, Lia Yeh, Richie Yeung, and Bob Coecke. Completeness for arbitrary finite dimensions of ZXW-calculus, a unifying calculus, April 2023. URL `http://arxiv.org/abs/2302.12135`.

Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. URL `https://api.semanticscholar.org/CorpusID:49313245`.

Marc'Aurelio Ranzato, Y-Lan Boureau, Sumit Chopra, and Yann LeCun. A unified energy-based framework for unsupervised learning. In Marina Meila and Xiaotong Shen (eds.), *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, volume 2 of *Proceedings of Machine Learning Research*, pp. 371–379, San Juan, Puerto Rico, 21–24 Mar 2007. PMLR. URL `https://proceedings.mlr.press/v2/ranzato07a.html`.

Oliver E. Richardson. Loss as the inconsistency of a probabilistic dependency graph: Choose your model, not your loss function. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera (eds.), *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pp. 2706–2735. PMLR, 28–30 Mar 2022. URL `https://proceedings.mlr.press/v151/richardson22b.html`.

Joseph Rocca. Understanding Variational Autoencoders (VAEs). https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73, March 2021.

Bernhard Scholkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5), 2021. ISSN 0018-9219.

Peter Selinger. A Survey of Graphical Languages for Monoidal Categories. In Bob Coecke (ed.), *New Structures for Physics*, pp. 289–355. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-12821-9. doi: 10.1007/978-3-642-12821-9_4. URL `https://doi.org/10.1007/978-3-642-12821-9_4`.

Lennert De Smet, Pedro Zuidberg Dos Martires, Robin Manhaeve, Giuseppe Marra, Angelika Kimmig, and Luc De Readt. Neural probabilistic logic programming in discrete-continuous domains. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 529–538. PMLR, July 2023.

Paweł Sobociński. Graphical Linear Algebra, June 2015. URL `https://graphicallinearalgebra.net/`.

Amos Storkey, Won-Min Seong, and Christopher K. I. Williams. Machine learning markets. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pp. 716–724, 2011.

Akhilesh Sudhakar, Bhargav Upadhyay, and Arjun Maheswaran. "Transforming" delete, retrieve, generate approach for controlled text style transfer. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3269–3279, 2019. URL `https://doi.org/10.48550/arXiv.1908.09368`.

Richard Sutton. The Bitter Lesson, 2019.

Juan Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, and Edgar A. Chavez-Urbiola. Loss Functions and Metrics in Deep Learning, September 2023. URL `https://doi.org/10.48550/arXiv.2307.0269`.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, New York, 1st edition edition, September 1998. ISBN 978-0-471-03003-4.

Xin Wang, Hong Chen, Si'ao Tang, Zihao Wu, and Wenwu Zhu. Disentangled Representation Learning, June 2024. Comment: Accepted by IEEE Transactions on Pattern Analysis and Machine Intelligence.

V. Wang-Maścianica. *String Diagrams for Text*. http://purl.org/dc/dcmitype/Text, University of Oxford, 2023.

Vincent Wang-Mascianica, Jonathon Liu, and Bob Coecke. Distilling Text into Circuits, January 2023. URL `http://arxiv.org/abs/2301.10595`.

Nicholas Watters, Loic Matthey, Sebastian Borgeaud, Rishabh Kabra, and Alexander Lerchner. Spriteworld: A flexible, configurable reinforcement learning environment, 2019. URL `https://github.com/deepmind/spriteworld/`.

Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989. doi: 10.1162/neco.1989.1.2.270. URL `https://doi.org/10.1162/neco.1989.1.2.270`.

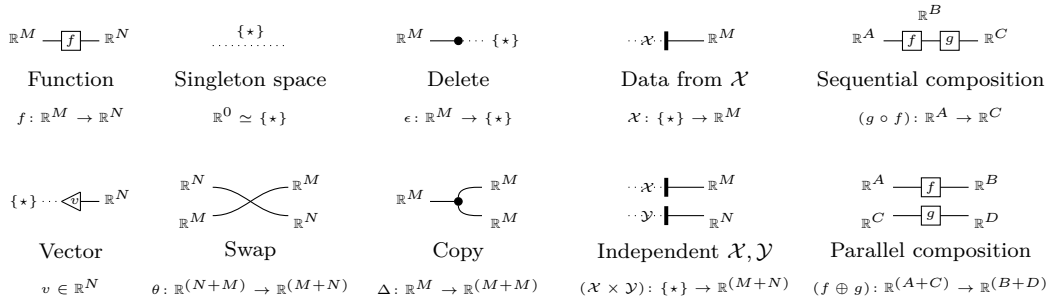
Sidi Wu, Yizi Chen, Samuel Mermet, Lorenz Hurni, Konrad Schindler, Nicolas Gonthier, and Loic Landrieu. Stegogan: Leveraging steganography for non-bijective image-to-image translation, March 2024. URL `https://doi.org/10.48550/arXiv.2403.20142`.

Donald Yau. Higher dimensional algebras via colored PROPs, September 2008. URL `https://doi.org/10.48550/arXiv.0809.2161`.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.

A String diagrams for tasks

Section 1 in the main text already introduces the working grammar of our diagrams. Here we collect the broader categorical background and semantics. String diagrams are a formal diagrammatic syntax that take semantics in symmetric monoidal categories, and they find usage in a broad variety of fields[4]. We reiterate the generators from Figure 1 for convenience:

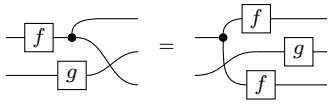


For conventional reasons that were not by our choice, vectors are depicted as triangular nodes with only output wires, reminiscent of bra–ket notation. (Co)associative (co)monoids, such as copy-delete and add-zero, are specially depicted as circular nodes as is common in applied category theory. In this work, encoders and decoders are sometimes depicted as "bottlenecking trapezia", as is common in ML, and distributional states are given their own notation as thick bars.

An attractive characteristic of string diagrams is that visually intuitive equivalences between information flows are guaranteed to correspond to symbolic derivations of behavioural equivalence: tedious algebraic proofs of equality between sequentially- and parallel-composite processes are suppressed and absorbed by (processive) isotopies of diagrams. In the diagrammatic syntax it is conventional to notate such isomorphisms as plain equalities. Interested readers are referred to (Selinger, 2011) for the relevant mathematical foundations.

$$
\begin{aligned}
&(\mathbf{1} \oplus \theta) \circ (\Delta \oplus g) \circ (f \oplus \mathbf{1}) \\
&\simeq (\mathbf{1} \oplus \theta) \circ (\mathbf{1} \oplus \mathbf{1} \oplus g) \circ (\Delta \oplus 1) \circ (f \oplus \mathbf{1}) && \text{[Identity, interchange]} \\
&\simeq (\mathbf{1} \oplus g \oplus \mathbf{1}) \circ (\mathbf{1} \oplus \theta) \circ (\Delta \oplus 1) \circ (f \oplus \mathbf{1}) && \text{[Braid naturality]} \\
&\simeq (\mathbf{1} \oplus g \oplus \mathbf{1}) \circ (\mathbf{1} \oplus \theta) \circ (f \oplus f \oplus 1) \circ (\Delta \oplus 1) && \text{[Copy naturality]} \\
&\simeq (\mathbf{1} \oplus g \oplus \mathbf{1}) \circ (f \oplus \mathbf{1} \oplus f) \circ (1 \oplus \theta) \circ (\Delta \oplus 1) && \text{[Braid naturality]}
\end{aligned}
$$

$\Leftrightarrow$



A.1 Categorical semantics of task diagrams

The functional effect of the construction below is to extend the category of continuous maps between Euclidean spaces with global elements that behave as probability distributions instead of points. We presume familiarity with symmetric monoidal categories and their graphical calculi (Selinger, 2011).

Let **CartSp** denote the coloured PROP (Yau, 2008) of continuous maps between Euclidean spaces, where the tensor product is the cartesian product — i.e. **CartSp** is cartesian monoidal.

Let **BorelStoch** denote the Markov category (Cho & Jacobs, 2019; Fritz, 2020) of stochastic kernels (Panangaden, 1999) between Borel-measurable spaces. Stochastic kernels in particular subsume the continuous maps between Euclidean spaces.

As a Markov category, in the terminology of (Fong & Spivak, 2019b), **BorelStoch** supplies cocommutative comonoids. By Fox's theorem (Fox, 1976b) cartesian monoidal categories are precisely those isomorphic to their own categories of cocommutative comonoids. Hence there is a (semicartesian) functorial embedding of **CartSp** into **BorelStoch** sending $\mathbb{R}^N$ to $\mathbb{R}^N$ (equipped with the usual Borel measure), and continuous maps

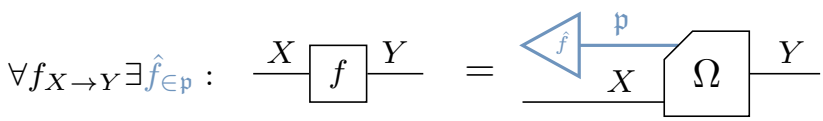
[4]Including linear and affine algebra (Sobociński, 2015; Bonchi et al., 2017; 2019), first order logic (Haydon & Sobociński, 2020), causal models (Lorenz & Tull, 2023; Jacobs et al., 2019), signal flow graphs (Bonchi et al., 2014), electrical circuits (Boisseau & Sobociński, 2022), game theory (Hedges, 2015), petri nets (Baez & Master, 2020), probability theory (Fritz et al., 2021), formal linguistics (Coecke et al., 2010; Wang-Mascianica et al., 2023; Wang-Maścianica, 2023), quantum theory (Coecke & Duncan, 2011; Coecke & Kissinger, 2017; Poór et al., 2023), and aspects of machine learning such as backpropagation (Cruttwell et al., 2022).

to deterministic continuous maps. We declare our semantics to be taken in the category to be generated by the image of this embedding along with the probability distributions $\mathcal{D} : \{\star\} \to \mathbb{R}^N$, where $\{\star\}$ is the singleton monoidal unit of **BorelStoch**.

A.2 Categorical semantics for idealised universal approximators

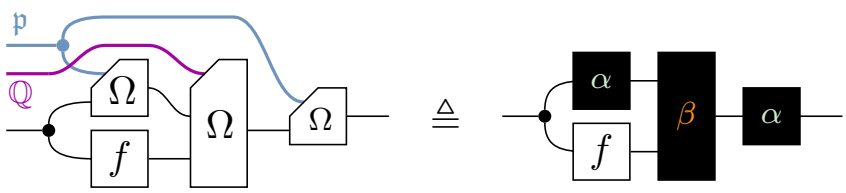
As we are concerned with behaviour, not implementation details, we idealise all neural networks as perfect universal approximators, which we may formulate string-diagrammatically in a monoidal closed category, borrowing evaluator-notation from (Pavlovic, 2013; 2023). In essence, we are assuming that architectures are sufficiently expressive to optimise whatever tasks we give them; in practice, the conditions under which architectures become universal approximators can be mild (Hornik et al., 1989), and the idealisation is increasingly true-in-practice in the contemporary context of increasing data and compute.

Definition A.1 (Learner)**.** Let X, Y denote input and output types. A process $\Omega : \mathfrak{p} \oplus X \to Y$ with parameters in $\mathrm{para}(\Omega) = \mathfrak{p} = \mathbb{R}^n$ (for sufficiently large n) is a *universal approximator* or *learner* when[5]:

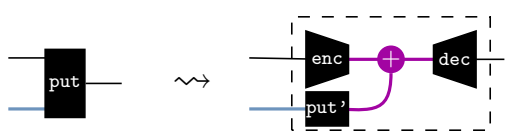


The parameter space could represent e.g. the phase space of weights and biases of a neural network.

Example A.2. By visual convention, we use colours to indicate different data types of wires. We depict processes with no free parameters as white boxes, and learners as black-boxes with variable labels to indicate distinct or shared parameters. The following composite process has one function f with no learnable parameters, and three neural nets: the two labelled α share a parameter in the space $\mathfrak{p}$, and the one labelled β takes a parameter in $\mathbb{Q}$. In this paper, we favour the shorthand on the right.



The universal approximation theorem, suitably idealised, manifests as the capacity for a black-box learner to be diagrammatically substituted for any other composite diagram with equal input and output, including those composites that contain other learners. For example, recall the linear `put` below, which may be viewed as substituting a particular composite of `put'`, `enc`, `dec`, and addition in place of `put`:



This ability is referred to in this work intermittently as *specialisation*, and as *expressive reduction* in (Khatri et al., 2024) where the concept first appeared. For the sake of completeness, we reproduce the relevant construction that gives category-theoretic semantics to universal approximators and specialisation below, along with standard definitions, with the authors' permission.

Definition A.3 (PROP)**.** A PROP is a strict symmetric monoidal category generated by a single object x: every object is of the form

$$\bigotimes^{n} x = \underbrace{x \otimes \cdots \otimes x}_{n}$$

PROPs may be generated by, and presented as *signatures* (Σ, E) consisting of generating morphisms Σ with arity and coarity in $\mathbb{N}$, and equations E relating symmetric monoidal composites of generators.

[5]We adapt the shape of the universal approximators to clearly indicate the parameters.

Definition A.4 (Coloured PROP)**.** A *multi-sorted* or *coloured* PROP with set of colours $\mathfrak{C}$ has a monoid of objects generated by $\mathfrak{C}$.

Definition A.5 (Cartesian PROP)**.** By Fox's theorem (Fox, 1976a), a cartesian PROP is one in which every object (wire) is equipped with a cocommutative comonoid (copy) with counit (delete) such that all morphisms in the category are comonoid cohomomorphisms.

Definition A.6 ((symmetric, unital) coloured operad)**.** Where $(\mathcal{V}, \boxtimes, J)$ is a symmetric monoidal category and $\mathfrak{C}$ denotes a set of *colours* c_i, a coloured operad $\mathcal{O}$ consists of:

- For each $n \in \mathbb{N}$ and each $(n+1)$-tuple $(c_1, \cdots, c_n; c)$, an object $\mathcal{O}(c_1, \cdots, c_n; n) \in \mathcal{V}$
- For each $c \in \mathfrak{C}$, a morphism $1_c : J \to \mathcal{O}(c; c)$ called the *identity of* c
- For each $(n+1)$-tuple $(c_1 \cdots c_n; c)$ and n other tuples $(d_1^1 \cdots d_{k_1}^1) \cdots (d_1^n \cdots d_{k_n}^n)$ a *composition morphism*
$$\mathcal{O}(c_1, \cdots, c_n; c) \boxtimes \mathcal{O}(d_1^1 \cdots d_{k_1}^1) \boxtimes \cdots \boxtimes \mathcal{O}(d_1^n \cdots d_{k_n}^n) \to \mathcal{O}(d_1^1 \cdots d_{k_1}^1 \cdots d_1^n \cdots d_{k_n}^n; c)$$
- for all $n \in \mathbb{N}$, all tuples of colours, and each permutation $\sigma \in S_n$ the symmmetric group on n, a morphism:
$$\sigma^* : \mathcal{O}(c_1 \cdots c_n; c) \to \mathcal{O}(c_{\sigma^*(1)} \cdots c_{\sigma^*(n)}; c)$$

The σ^* must represent S_n, and composition must satisfy associativity and unitality in a S_n-invariant manner.

Construction A.7 (Hom-Operad of coloured PROP)**.** Where $(\mathcal{P}, \otimes, I)$ is a coloured PROP with colours $\mathfrak{C}_{\mathcal{P}}$, we construct $\mathcal{O}_{\mathcal{P}}$, the *hom-operad* of $\mathcal{P}$. We do so in two stages, by first defining an ambient operad, and then restricting to the operad obtained by a collection of generators. Let the ambient symmetric monoidal category be $(\mathbf{Set}, \times, \{\star\})$. Let the colours $\mathfrak{C}_{\mathcal{O}}$ be the set of all tuples $(\mathbf{A}, \mathbf{B})$, each denoting a pair of tuples $(A_1 \otimes A_n, B_1 \otimes B_n)$ of $A_i, B_i \in \mathfrak{C}_{\mathcal{P}}$.

- The tuple $\big((\mathbf{A}^1, \mathbf{B}^1) \cdots (\mathbf{A}^n, \mathbf{B}^n); (\mathbf{A}, \mathbf{B})\big)$ is assigned the set $[\mathcal{P}(\mathbf{A}^1, \mathbf{B}^1) \times \cdots \times \mathcal{P}(\mathbf{A}^n, \mathbf{B}^n) \to \mathcal{P}(\mathbf{A}, \mathbf{B})] \in \mathbf{Set}$; the set of all *generated* functions from the product of homsets $\mathcal{P}(\mathbf{A}^i, \mathbf{B}^i)$ to the homset $\mathcal{P}(\mathbf{A}, \mathbf{B})$.
- $1_{(\mathbf{A},\mathbf{B})} : \{\star\} \to [\mathcal{P}(\mathbf{A}, \mathbf{B}) \to \mathcal{P}(\mathbf{A}, \mathbf{B})]$ is the identity functional that maps $f : \mathbf{A} \to \mathbf{B}$ in $\mathcal{P}(\mathbf{A}, \mathbf{B})$ to itself.
- The composition operations correspond to function composition in $\mathbf{Set}$, where $[X \to Y] \times [Y \to Z] \to [X \to Z]$ sends $(f_{:X \to Y}, g_{:Y \to Z}) \mapsto (g \circ f)_{:X \to Z}$; appropriately generalised to the multi-argument case. The permutations are similarly defined, inheriting their coherence conditions from the commutativity isomorphisms of the categorical product $\times$.

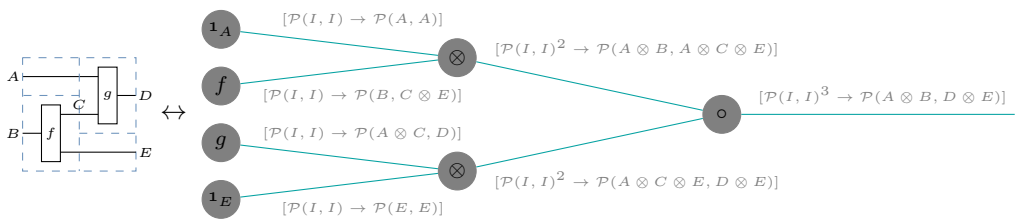
The generators are:

- For every $f \in \mathcal{P}(\mathbf{A}, \mathbf{B})$ that is a generator of $\mathcal{P}$, define a corresponding generator of type $\{\star\} \to [\mathcal{P}(I, I) \to \mathcal{P}(\mathbf{A}, \mathbf{B})]$, which is the functional $\big(- \mapsto (f \otimes -)\big)$ that sends endomorphisms of the monoidal unit of $\mathcal{P}$ to their tensor with f, viewed as an element of the set $[\mathcal{P}(I, I) \to \mathcal{P}(\mathbf{A}, \mathbf{B})]$.
- For every pair of tuples $\big((\mathbf{X}^1, \mathbf{Y}^1) \cdots (\mathbf{X}^m, \mathbf{Y}^m); (\mathbf{A}, \mathbf{B})\big)$ and $\big((\mathbf{J}^1, \mathbf{K}^1) \cdots (\mathbf{J}^n, \mathbf{K}^n); (\mathbf{B}, \mathbf{C})\big)$ in $\mathfrak{C}_{\mathcal{O}}$, a corresponding *sequential composition* operation of type:
$$[\prod_{i \leqslant m} \mathcal{P}(\mathbf{X}^i, \mathbf{Y}^i) \to \mathcal{P}(\mathbf{A}, \mathbf{B})] \times [\prod_{j \leqslant n} \mathcal{P}(\mathbf{J}^j, \mathbf{K}^j) \to \mathcal{P}(\mathbf{B}, \mathbf{C})]$$
$$\to [\big(\prod_{i \leqslant m} \mathcal{P}(\mathbf{X}^i, \mathbf{Y}^i) \times \prod_{j \leqslant n} \mathcal{P}(\mathbf{J}^j, \mathbf{K}^j)\big) \to \mathcal{P}(\mathbf{A}, \mathbf{C})]$$
Which maps pairs of functionals $(F_{:\prod_{i \leqslant m} \mathcal{P}(\mathbf{X}^i, \mathbf{Y}^i) \to \mathcal{P}(\mathbf{A}, \mathbf{B})}, G_{:\prod_{j \leqslant n} \mathcal{P}(\mathbf{J}^j, \mathbf{K}^j) \to \mathcal{P}(\mathbf{B}, \mathbf{C})})$ to the functional which sends $p^i : \mathbf{X}^i \to \mathbf{Y}^i$ and $q^j : \mathbf{X}^j \to \mathbf{Y}^j$ to $G(p_1 \cdots p_m) \circ F(q_1 \cdots q_n)$.

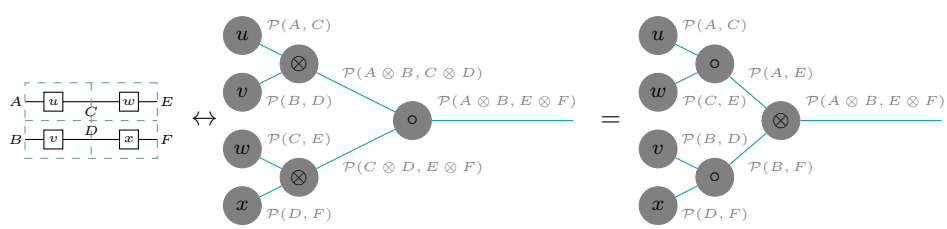
- An analogous *parallel composition* for every pair of tuples, which sends pairs of functionals (F, G) to $G(p_1 \cdots p_m) \otimes F(q_1 \cdots q_n)$.

Remark A.8. For technical reasons involving scalars (the endomorphisms of the monoidal unit), this construction only works in semicartesian settings, i.e. where the monoidal unit is also terminal, but that is sufficiently general to admit our use cases, which are primarily in cartesian monoidal settings (Fox, 1976a) and semicartesian Markov categories for probabilistic settings (nLab authors, 2024b).

Example A.9. Construction A.7 can be thought of as bridging diagrams with their specific algebraic descriptions using just the basic constructors $\circ, \otimes$; the hom-operad (when notated suggestively in the usual tree-notation, found e.g. in Markl et al. (2007)) plays the role of the syntactic tree of $\circ, \otimes$ operators. For instance, given the composite morphism $(g \otimes 1_E) \circ (1_A \otimes f)$ in PROP $\mathcal{P}$, the corresponding diagram and operad-state in $\mathcal{O}_{\mathcal{P}}$ is:



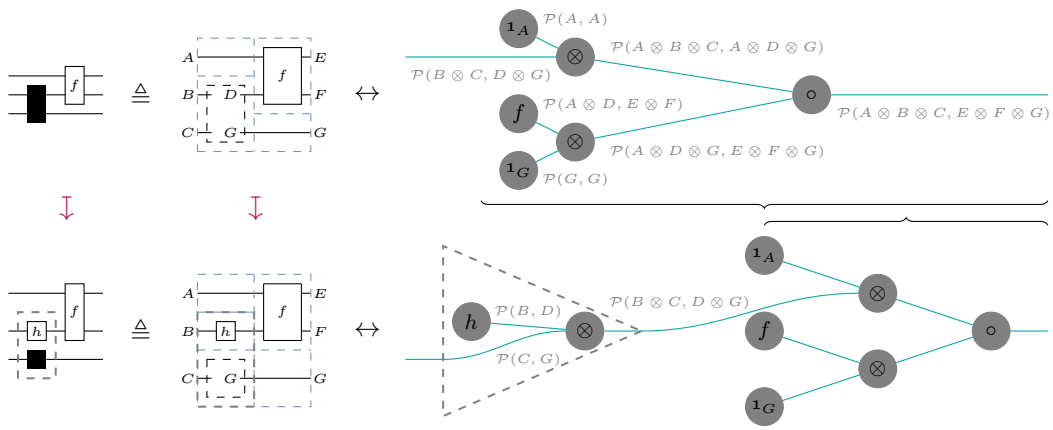
Since the PROPs **CartSp** and its free tensoring are cartesian, $\mathcal{P}(I, I)$ is a singleton containing only the identity of the monoidal unit, so in the settings we are concerned with, we may simplify colours of the form $[\mathcal{P}(I, I)^N \to \mathcal{P}(\mathbf{A}, \mathbf{B})]$ to just $\mathcal{P}(\mathbf{A}, \mathbf{B})$, and operad-states $\{\star\} \to \mathcal{P}(\mathbf{A}, \mathbf{B})$ are in bijective correpondence with morphisms $f : \mathbf{A} \to \mathbf{B}$ of $\mathcal{P}$; the fact that all $f : \mathbf{A} \to \mathbf{B}$ are representable as operad states follows by construction, since any f in $\mathcal{P}$ is by definition expressible in terms of the generators of $\mathcal{P}$, and sequential and parallel composition $\circ, \otimes$. As we assume homsets are already quotiented by the equational theory of $\mathcal{P}$ and the symmetric monoidal coherences, our operadic representations inherit them: for example, we obtain interchange equalities such as the one below for free:



Definition A.10 (Universal approximators and specialisation)**.** A morphism of a coloured PROP $\mathcal{P}$ of type $(\mathbf{A}, \mathbf{B})$ containing universal approximators as black-boxes of types $\mathbf{A}^{i \leqslant n} \to \mathbf{B}^{i \leqslant n}$ is a morphism $\big((\mathbf{A}^1, \mathbf{B}^1) \cdots (\mathbf{A}^n, \mathbf{B}^n); (\mathbf{A}, \mathbf{B})\big)$ of $\mathcal{O}_{\mathcal{P}}$, and by construction, vice versa. Specialisation corresponds to precomposition in $\mathcal{O}_{\mathcal{P}}$.

Example A.11. The inputs of open morphisms in $\mathcal{O}_{\mathcal{P}}$ correspond to "typed holes", and operadic precomposition corresponds to "filling holes", with contents that may themselves also contain typed holes. This precisely formalises the intuition that expressive reductions correspond to the ability of a universal approximator to

simulate anything, including composites containing other universal approximators.



Remark A.12. The extension of the current theory to accommodate parameter sharing between universal approximators is conceptually straightforward but technically involved. Parameter sharing corresponds to the ability to reuse – i.e. copy – data between open wires in the operad $\mathcal{O}_{\mathcal{P}}$, which amounts to having a cartesian operad.

B `Strong manipulation`

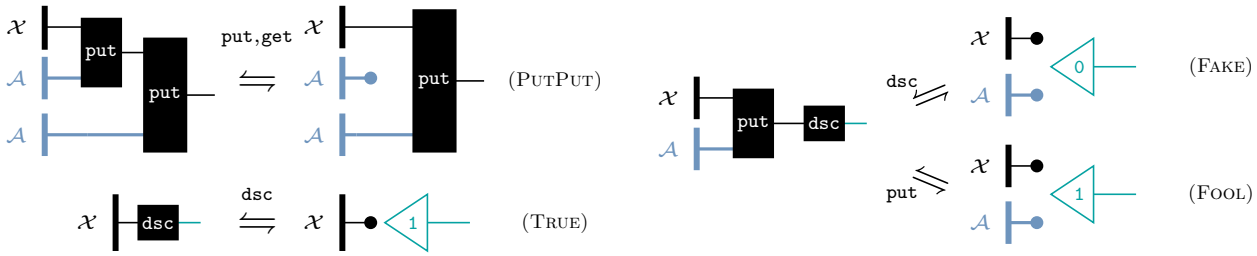
The basic `manipulation` admits pathological counterexamples, which we may block by imposing additional tasks as regularisation terms. This section further illustrates a form of legalistic thinking using tasks: by thinking of ways that "noncooperative" or "naughty" learners might seek to satisfy tasks without exhibiting the behaviour that the modeller desires. By identifying these counterexamples and constructing additional tasks that block them, the modeller may iteratively improve the behaviour of the model. For illustration, consider the following examples of pathological behaviour that satisfy basic `manipulation`, again in the setting of editing the colour of a shape.

Example B.1 (Flipping)**.** Consider a `put` that changes the colour of a shape as desired, but then vertically flips the shape. If the classifier `get` is insensitive to the position and orientation of the shape, then Classify, PutGet, and GetPut are satisfied. Moreover, since a vertical flip is its own inverse, composing two `put`s as in the Undoability task will not detect this aberration. Speaking in more general terms, if there are properties that `get` is insensitive to, there must be additional guardrails to ensure that `put` preserves these other properties as the identity, rather than one of potentially very many self-inverse symmetries.

Example B.2 (Adversarial decorations)**.** While the classifier `get` may be perfect in-distribution, there are no guarantees about its behaviour out of distribution, for example, when given images with multiple shapes, where it might only classify the leftmost shape. So, it is possible that `put` learns to make edits that take the resulting image out-of-distribution: for example, by adding a red circle next to a blue square to fool the classifier into outputting "red". This would satisfy Classify, PutGet, and GetPut. If the `put` can recognise and undo its own decorations, then Undoability will also be satisfied. Speaking more generally, we require additional guardrails to ensure that `put` returns something in-distribution.

The `strong manipulation` adds additional tasks to `manipulation`. A `strong manipulator` has to satisfy the original four tasks plus the following four:

Task B.3 (`strong manipulation` add-ons)**.** The `strong manipulation` task consists of all the tasks contained in `manipulation`, we well as:



The PutPut task (which is strictly stronger than Undoability in that it is algebraically implied) says that the effect of `put`ting twice is the same as discarding the effect of the first edit and only keeping the last edit. In conjunction with PutGet and GetPut, this creates what is known in the literature as a *very well-behaved lens*, which blocks Example B.1 and similar modifications of the data `get` is insensitive to.

The True, Fake, and Fool tasks introduce a discriminator component `dsc`, which forms a `GAN` pattern with respect to `put` as the generator. When well-trained, this forces the outputs of `put` to lie in-distribution. As in general there are no algebraic or equational laws that characterise arbitrary distributions of data, using `GAN`s in this way is a generic recipe for shaping outputs of generators to behave well in-distribution.

Remark B.4 (Why basic `manipulation` is preferable in practice)**.** We have observed informally that conditions such as those in basic `manipulation` where learners are cooperative and there are only learners on the LHS appear to be more stable during training. We suggest a sketch reason why: in the tasks of `strong manipulation`, PutPut has `put` occur on both the LHS and RHS, which establishes a nontrivial dependence on the current position on parameter-space of `put` in the process of finding a solution. Similarly, the `GAN` rules of `strong manipulation` establishes adversarial mutual dependencies in the parameters of `dsc` and `put`. Conceptually, these dependencies create dynamical systems on the paths that the learners take over the course of training in parameter-space, which may for instance include stable orbits and chaotic behaviour, and may be highly sensitive to initial conditions. A further elaboration of "static" versus "dynamic" tasks in tandem with the ability to express equivalent tasks is potentially useful for creating train-stable models with equivalent behaviour, but this is beyond the scope of this paper, and left for future work.

C Experiment Details

C.1 Stack

This experiment uses an autoencoder architecture for processing 32×32 RGB images, with a latent space size of 16 dimensions. The encoder consists of four convolutional layers, each with 64 channels and a channel multiplier of 1. Additionally, a stack of 64 latent features is processed through an MLP with 256 hidden units. Training is performed on a GPU, with a batch size of 64, learning rate of 1×10^{-4}, weight decay of 1×10^{-2}, and gradient clipping at 1. The model trains for 100,000 steps, logging every 10,000 steps. Input images are converted to tensors and scaled to floating-point precision, and a fixed random seed of 0 ensures reproducibility. The stack space S has to be n times larger than the latent space **Lat** for the stack to be able to hold up to n items.

C.2 Spriteworld

For the Spriteworld experiment, we procedurally generate 32x32 images containing a single shape with the following attributes:

Attributes	*Possible Values*
Shape	{ Ellipse, Rectangle, Triangle }
Hue	{ Red : 0±8, Green : 85±8, Blue : 170±8 }
Saturation	64-255
Value	64-255
Background Color	Black
Width & Height	5-27
X & Y position	5-27

Only the first two attributes, shape and hue, are changed in the `manipulation` task, but all unchanged properties are intended to be preserved by the transformation. We use an autoencoder with a CNN/DCNN architecture to embed each image into a latent space:

Parameter	*Value*
Latent Size	32
Layers	4
Hidden Channels	64
Kernel Size	5x5
Stride	2
Activation Function	LeakyReLU(0.1) followed by BatchNorm

We train separate `get`/`put` models for each of the three concepts: shape, colour, blue-circleness. Each model uses the encoder of the autoencoder to embed input images into latent space, and only sees the labels for the particular attribute it is manipulating.

For shape and colour, the `get` model uses a linear classifier from the latent space (of size 32) to 3 output logit values, one for each possible value. The `put` model maps a one-hot vector of the input value to a vector in latent space that is added to the embedding. This new embedding is then decoded by the autoencoder.

For blue-circleness, the `get` model uses a linear classifier from the latent space to a single output value from zero to one (we do not use a sigmoid output layer to restrict the output). The `put` model uses a complement of size 8. It concatenates the one-hot value vector with the image embedding and the complement vector (using a default trainable complement vector if one is not provided) and passes that through a linear layer to get a new embedding vector (which is then decoded) and complement vector.

In total, these models contain 644,130 parameters. All models are trained simultaneously according to the `autoencoding` and `manipulation` rules, along with PutPut. At each step, a batch of images is generated, along with four batches of random values, containing random labels for the shape and colour, and random real numbers for the blue-circleness, uniformly sampled from $[-0.1, 1.1]$ and then clamped to $[0, 1]$. The loss function is a weighted sum of the losses from each atomic task in order to balance the signal from the image loss with the signal from the classifier loss:

Hyper-parameter	*Value*
Steps	100,000
Batch Size	512
Optimiser	AdamW
Learning Rate	10^{-3}
Weight Decay	10^{-2}
Gradient Clipping	1 (element-wise)
Image Loss	$L_2 + 0.25 \cdot L_1$
Discrete Value Loss	Binary cross-entropy
Continuous Value Loss	Mean squared error
Seed	0

Task	*Weight*
autoencoding	100
GetPut	1
PutPut	1
Undo	10
PutGet	
(blue-circleness)	10
(shape and colour)	1
Classification	
(blue-circleness)	10
(shape and colour)	1

C.3 Faces

For the faces experiment, we use the *CelebFaces Attributes* dataset (Liu et al., 2015), with an off-the-shelf data augmentation method called TrivialAugment (Müller & Hutter, 2021). Again, we use an autoencoder with a CNN/DCNN architecture to embed each image:

Parameter	*Value*
Latent Size	128
Layers	5
Hidden Channels	8, 16, 32, 64, 128
Kernel Size	5
Stride	2
Activation Function	LeakyReLU(0.1) followed by BatchNorm

We train linear `get`/`put` models for the binary concept of "Smiling", resulting in a total of 1,071,749 parameters. The loss function is a weighted sum of the losses from each atomic task:

Hyper-parameter	*Value*
Steps	100,000
Batch Size	64
Optimiser	AdamW
Learning Rate	10^{-3}
Weight Decay	10^{-2}
Gradient Clipping	1 (element-wise)
Image Loss	$L_2 + 0.2 \cdot L_1 + SSIM$
Value Loss	Binary cross-entropy
Seed	0

Task	*Weight*
AUTOENCODING	10
GETPUT	1
PUTPUT	1
UNDO	1
PUTGET	1
CLASSIFICATION	1

C.4 MNIST

We trained the `manipulation` task on the MNIST dataset, using the digit label as the property. The `get` operated directly on images, while `put` was trained to act on the latent space of an autoencoder, as in option (1) of Section 4.3. The images are input as 28×28 matrices, flattened to 784-dimensional vectors, and the labels are provided as 10-dimensional vectors with one-hot encoding. All of the components were structured as multilayer perceptrons. The hyperparameters are given below:

	`enc`	`dec`	`put`	`get`
Input Dimension	$784 = 28 \times 28$	32	$42 = 32 + 10$	$784 = 28 \times 28$
Output Dimension	32	$784 = 28 \times 28$	32	10
Hidden Dimensions	$\{128, 128, 64\}$	$\{64, 128, 128\}$	$\{128, 128, 128\}$	$\{64, 64\}$
Hidden Activations	ReLU	ReLU	ReLU	ReLU
Final Activation	Sigmoid	Sigmoid	Sigmoid	Softmax

With these architectural components, we trained four tasks: (a) training `get` supervised, (b) training `get` given pre-trained `put`, `enc`, and `dec`, (c) training `put` given pre-trained `get'`, `enc`, and `dec`, and (d) training `get` to match a previous `get'`. These were trained using the `manipulation` rules, as well as PUTPUT, and additional regularization term we denote as ENTROPY. The loss function of ENTROPY is given by

$$\mathcal{L}_{\text{ENTROPY}} = \mathbb{E}[H(\texttt{get}(\texttt{enc}(x)))] - H(\mathbb{E}[\texttt{get}(\texttt{enc}(x))])$$

where x is a batch of input images, $H(\cdot)$ is the entropy of a categorical distribution, and the expectation is approximated by the mean over each batch. The idea behind ENTROPY is to encourage the output of `get`

to be well-distributed across labels (by maximizing the entropy of the mean distribution) but to be sure of each label (by minimizing the entropy for each specific input). For task (d), labels were generated for the CLASSIFY rule using `get'`. Each task is trained by minimizing a weighted linear combination of the rules. We give the hyperparameters, rule weights, and loss functions for each of these below.

	(a)		(b)		(c)		(d)	
Optimizer	Adam		Adam		Adam		Adam	
Learning Rate	0.001		0.001		0.0001		0.001	
Epochs	20		20		20		20	
	Weight	*Loss*	*Weight*	*Loss*	*Weight*	*Loss*	*Weight*	*Loss*
CLASSIFY	1	CE	–	–	–	–	1	CE
PUTGET	–	–	10	CE	10	CE	–	–
GETPUT	–	–	10	L2	10	L2	–	–
PUTPUT	–	–	10	L2	10	L2	–	–
UNDOABILITY	–	–	10	L2	10	L2	–	–
ENTROPY	–	–	1	$\mathcal{L}_{\text{ENTROPY}}$	–	–	–	–

We also have an additional task (e) of training `enc` and `dec` unsupervised. This was done using the Adam optimizer, with a learning rate of 0.001 for 80 epochs. The reconstruction loss was given by

$$\mathcal{L}(x, \hat{x}) = \text{L2}(x, \hat{x}) + (1 - \text{SSIM}(x, \hat{x}))$$

where SSIM is the structure similarity image metric.

To produce Figure 4, an `enc`, `dec`, `put`, and `get` were trained using (a) → (e) → (c), followed by training (c) for an additional 40 epochs. A slightly larger (but still MLP-based) model, where both `put` and `get` act on the latent space of the autoencoder, was used to achieve better visual quality. The hyperparameters are detailed below:

	`enc`	`dec`	`put`	`get`
Input Dimension	$784 = 28 \times 28$	32	$42 = 32 + 10$	32
Output Dimension	32	$784 = 28 \times 28$	32	10
Hidden Dimensions	{128, 128, 128, 32, 32}	{32, 32, 128, 128, 128}	$\{256, 256\}$	{256}
Hidden Activations	ReLU	ReLU	ReLU	ReLU
Final Activation	Sigmoid	Sigmoid	Sigmoid	Softmax

`enc`, `dec`, and `put` were used to manipulate six examples picked from the dataset, putting each of the ten classes onto each example. The examples were cherrypicked to provide maximum stylistic contrast across the sample but were not selected for maximum style transfer accuracy - a similar level was observed across the entire dataset. Code for all of these models, as well as the training schedules of tasks (a)-(e), are provided in the supplementary material.

D Examining `Manipulation` in complex domains

D.1 `Manipulation` for text sentiment

We attempted to fine-tune an extant strong solution for text-sentiment modification by additionally imposing the constraints of `manipulation` on top of the original objective function. Our findings suggest that additionally imposing the constraints of `manipulation` on architectures that are already performant does not make an appreciable difference (Table 1). We pretrained the Blind Generative Style Transformer (**B-GST**) model of (Sudhakar et al., 2019) which takes in the non-stylistic components of a sentence and the target

sentiment, and outputs the sentence generated in the target style. This was done until we achieved baselines higher than the original ones reported by the authors of the model. Afterwards, we continued training under three different conditions: (1) resuming training with only the original objective, (2) only using objective functions from `manipulation`, and (3) using both. For (1) and (3), we reused the same objective in the original paper. In all experiments, we used the YELP dataset used by (Li et al., 2018), reusing the same train-dev-test split they used. It consists of 270K positive and 180K negative sentences for the training set, 2000 sentences each for the dev set, and 500 sentences each for the test set. Furthermore, we used the human gold standard references they provided for their test set. **B-GST** uses a sequence length of 512, 12 attention blocks each with 12 attention heads. We used 768-dimensional internal states (keys, queries, values, word embeddings, positional embeddings). We tokenized the input text using Byte-Pair Encoding (BPE).

We used the same input autoencoding and output decoding used in (Sudhakar et al., 2019) across all experiments. For the `get` of the `manipulation` task, we used the PyTorch version of the pretrained Transformer by HuggingFace, which uses the OpenAI GPT model pretrained by (Radford & Narasimhan, 2018) on the BookCorpus dataset which contains over 7000 books with approximately 800M words. We trained it on a sentiment classification task using the YELP dataset reaching 98% accuracy on the test set. The `get` was fixed for the entire duration of training conditions (2) and (3) above. For the `put` of `manipulation`, we used the **B-GST** model to generate text with a specified sentiment. This was a computational bottleneck for training conditions (2) and (3) as autoregressive decoding is required to generate model inputs for PutGet and Undoability in `manipulation`. We used 'teacher forcing' or 'guided approach' (Bengio et al., 2015; Williams & Zipser, 1989) whenever we computed the reconstruction loss of `put`. Additionally, for training conditions (2) and (3), we only used PutGet, GetPut, and Undoability from `manipulation`. We used a weighted sum of the losses computed for each of these and the original reconstruction loss if present - the weights can be considered as training hyperparameters. For (2), we used (PutGet=5, GetPut=20, Undoability=20), while for (3), we used (PutGet=5, GetPut=10, Undoability=25, B-GST=30). Code for all of the models, training schedules, and hyperparameter values for training conditions (1)-(3) are also provided in the supplementary material.

Model	GLEU	BLEU_{SRC}	BLEU_{REF}	ACC (fasttext)
`B-GST`-pretrained	**11.869**	74.563	52.770	84.6
`B-GST`-only	11.426	**74.876**	52.549	**85.7**
`manipulation`-only	11.712	74.428	52.646	84.1
`B-GST+manipulation`	11.338	74.608	**52.836**	85.1
`Human Reference`	100.00	58.158	100.00	67.6

Table 1: We pretrained the Blind Generative Style Transformer (**B-GST**) model (Sudhakar et al., 2019) based on the `Delete-Retrieve-Generate` (Li et al., 2018) framework for sentiment modification until we recovered higher baselines than reported by the authors of the model (GLEU=11.6, BLEU_{SRC}=71.0), and we continued training in three different conditions: (1) keeping the original objective, (2) only using objective functions obtained from `manipulation`, and (3) using both. We report no statistically significant differences in scores, even under continued training. The table reports the results of training conditions (1) and (2) for an additional epoch, and (3) for two epochs.

D.2 Characterising `Manipulation` as generative classification

Training `manipulation` autoregressively (e.g. when instantiating the learners as transformers for sequential data) is slow due to autoregressing twice for PutPut and `Undoability`. Moreover, our attempts to autoregressively manipulate the sentiment of IMDB reviews often resulted in a form of posterior collapse where `put` ignored the attribute and behaved as the identity function on text. Conceptually, this is because the identity function satisfies GetPut, PutPut and Undoability, and while the identity fails on PutGet, failing on one component of the combined loss function does not provide a strong enough incentive to move away from the identity in parameter-space.

Notably, these shortcomings mirror that of VAEs, which also suffer from posterior collapse (Bond-Taylor et al., 2022) in highly structured domains such as video (Babaeizadeh et al., 2018) and text (Bowman et al., 2016). This suggested to us that `put`s may be generative classifiers, which could potentially explain why mode collapse was occurring in complex domains. Borrowing terminology from (Ng & Jordan, 2001), classifiers are *discriminative* if they seek to learn the conditional distribution $p(a|d)$ of attributes given data (as in the `classification` pattern), and otherwise they are *generative* if they seek to learn the joint distribution $p(d, a)$ (as, for example, a VAE). The tradeoffs between the two types are well studied, e.g. performance-wise, generative models may converge faster with limited data, but discriminative models often achieve lower asymptotic error, and it is well known that learning generative models is harder (Vapnik, 1998).

`Manipulation` cannot be viewed directly as a generative classifier, as the `put` admits extraneous conditionalisations on a reference and a target attribute. So we resorted to an empirical comparison of `manipulation` and VAEs as a known generative classifier; specifically, we compared their ability to approximate the Bayesian inverse, and we found their performance comparable. To demonstrate this, we tried three ways to train an informationally identical `cls`$'$ given an initial `cls`, provided access to unlabelled data to obtain a distribution of pairs $(d, \texttt{cls}(d))$ by: (a) directly training `cls` by `classification`, (b) training a generative classifier, in our case a VAE, and (c) training `manipulation` around `cls`-as-`get` to obtain a `put`, and then train `cls`$'$ to satisfy the tasks of `manipulation` except for Classify. Repeating this process several times, we would expect to see some loss of accuracy due to imperfect Bayesian inversion. Indeed, we see in Figure 9 that (b) and (c) have similar decays in accuracy, indicating that `manipulation` and VAEs have similar performance in this case. This experiment was performed using the trained components obtained from the MNIST experiment (Section C.4), and in addition to the tasks (a-e) we (f) trained a VAE to learn the joint distribution of images and labels produced by `get'`, and we (g) trained a `get` supervised using labels and images generated from the VAE. The VAE encoder and decoder are also based on multilayer perceptrons - each is comprised of an MLP trunk and two linear heads for generating the means and log-variances of the latent space, or the image and labels, respectively. The latent space is comprised of independent normally distributed variables as in (Kingma & Welling, 2022), and is sampled using the standard reparameterization trick. The hyperparameters of the architecture are given below:

	VAE Encoder	VAE Decoder
Input Dimension	$794 = 28 \times 28 + 10$	32
Hidden Dimensions	$\{128, 128, 64\}$	$\{64, 128, 128\}$
Hidden Activations	ReLU	ReLU
Head 1 Dimension	32	$784 = 28 \times 28$
Head 1 Activation	–	Sigmoid
Head 2 Dimension	32	10
Head 2 Activation	–	Softmax

Three loss functions were used for tasks (f) and (g) — the reconstruction loss of the autoencoder, which can be separated into a label loss and an image loss, the K-L divergence regularization term $\mathcal{L}_{KL}$ of the VAE (Kingma & Welling, 2022), and the Classify loss of `get`. The training hyperparameters are given as follows:

	(f)		(g)	
Optimizer	Adam		Adam	
Learning Rate	0.001		0.001	
Epochs	40		20	
	Weight	*Loss*	*Weight*	*Loss*
CLASSIFY	–	–	1	CE
Image Reconstruction	100	L2	–	–
Label Reconstruction	1	CE	–	–
K-L Divergence	0.5	$\mathcal{L}_{KL}$	–	–

In order to evaluate the three methods, we trained the tasks in the following order. At each step, the component being trained (e.g. `get`, `put`, etc) was initialized randomly (the previous weights were discarded). Measurements of the test accuracy were made after each (a), (b), (d), or (g) training run, and used to produce Figure 9.

$$\texttt{get} \rightarrow \texttt{get'} \implies \text{(a)} \rightarrow \text{(d)} \rightarrow \text{(d)} \rightarrow \cdots \rightarrow \text{(d)}$$
$$\texttt{get} \rightarrow \texttt{put} \rightarrow \texttt{get'} \implies \text{(a)} \rightarrow \text{(e)} \rightarrow \text{(c)} \rightarrow \text{(b)} \rightarrow \text{(e)} \rightarrow \text{(c)} \rightarrow \text{(b)} \rightarrow \cdots \rightarrow \text{(b)}$$
$$\texttt{get} \rightarrow \texttt{VAE} \rightarrow \texttt{get'} \implies \text{(a)} \rightarrow \text{(f)} \rightarrow \text{(g)} \rightarrow \text{(f)} \rightarrow \text{(g)} \rightarrow \cdots \rightarrow \text{(g)}$$

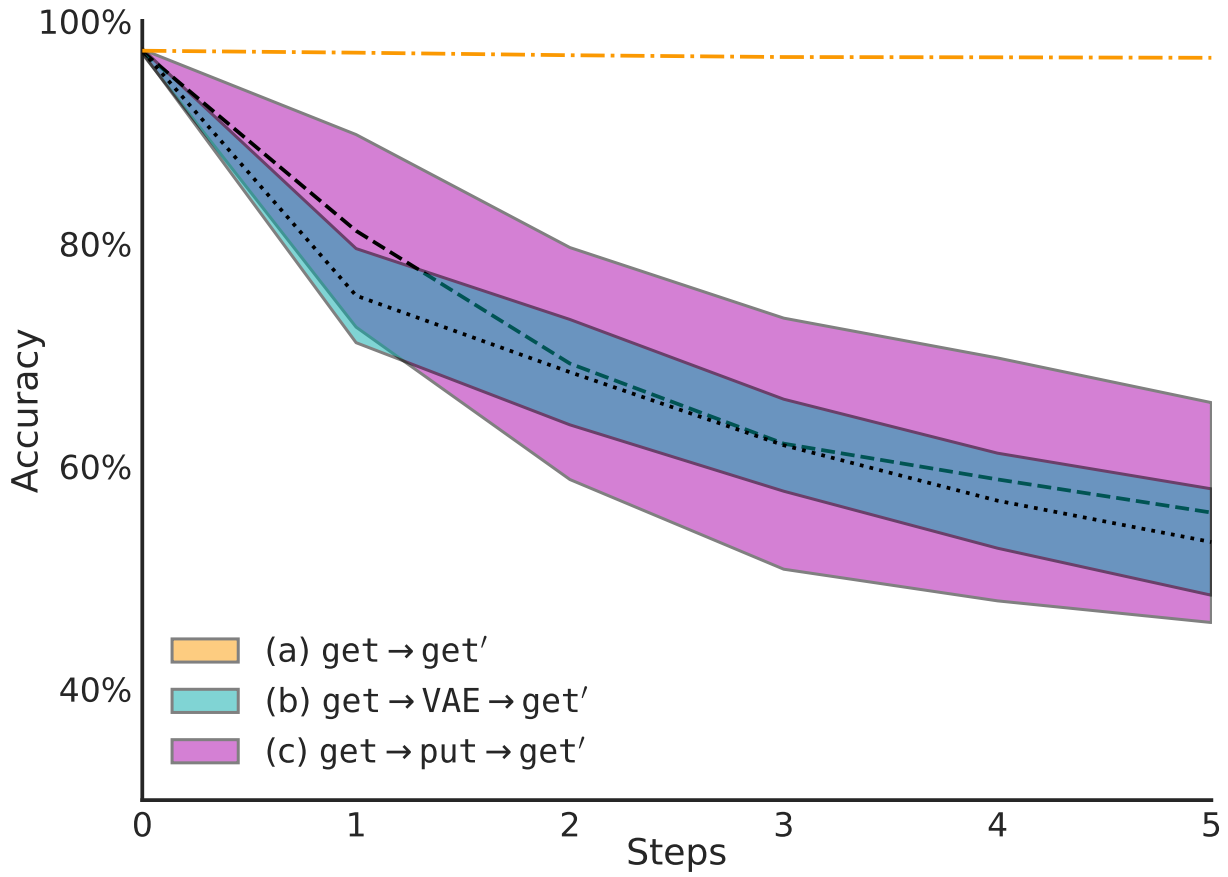


Figure 9: We tried three ways to distil an informationally identical `cls′` given an initial `cls` - depicted are the results of training successive MNIST classifiers using methods (a), (b) and (c) given above. 'steps' refers to the number of times this process was repeated. We observe that the degradation of accuracy is approximately the same for both (b) and (c), which we consider evidence that `manipulator`s and `VAE`s have similar performance characteristics. Both models had roughly the same number of parameters (300K) and were based on the same MLP architecture. We ran 20 repetitions of each method, the shaded regions represent one standard deviation (method (a) had a standard deviation of less than 1%).