ChatGPT

Comparative Study of Multimodal Representations in Deep Learning – Implementation Blueprint

Introduction and Objectives

This implementation blueprint outlines a **comparative study of multimodal representations** using multiple datasets and fusion strategies. The thesis will follow **Path A**, employing separate datasets (FashionAI, MovieLens, Amazon Reviews) to cover different modality combinations (image-text, image-text-graph-numeric, text-metadata). Key objectives include:

- **Benchmarking Fusion Paradigms:** Evaluate three modeling paradigms – **classical fusion** (early vs. late fusion), **transformer-based fusion** (e.g. ViLBERT, CLIP), and **generative fusion** via a **Multimodal VAE** – on each dataset. This provides a broad comparison of how each approach learns joint representations.
- **Hybrid Fusion Model:** Propose and implement a **hybrid model** that feeds the latent vector *z* from a multimodal VAE into a downstream classifier. This combines generative and discriminative training, aiming to leverage the VAE's latent space for improved supervised performance [1].
- **Comprehensive Evaluation:** Assess models on (1) **predictive performance** (e.g. classification accuracy or rating prediction error), (2) **semantic alignment** across modalities (e.g. cross-modal retrieval tasks to see if modalities align in latent space), and (3) **interpretability** (e.g. visualize attention maps, Grad-CAM heatmaps, and examine latent factor disentanglement).
- **Reproducible Implementation:** Use **PyTorch Lightning** and **Hydra** for a modular, configurable training pipeline. Lightning abstracts the training loop for cleaner code and easier debugging, while Hydra manages experiment configurations for reproducibility and easy hyperparameter tuning [2] [3].

Novelty: This project's novelty lies in **systematically comparing diverse multimodal fusion techniques** on multiple data domains and proposing a **hybrid generative-discriminative model**. Unlike prior works focusing on a single approach, we will illuminate trade-offs between early/late fusion vs. transformer models vs. VAEs under consistent settings. Moreover, integrating a VAE's latent representation into a predictive model (hybrid fusion) is a relatively under-explored idea [4] in multimodal learning, which we will explore along with an extensive interpretability analysis. By analyzing *semantic alignment* (cross-modal retrieval performance and shared latent space structure) and *interpretability* (e.g. which image regions or words influence decisions via attention/Grad-CAM), the study provides deeper insights beyond accuracy alone. This clarity and breadth of analysis, combined with the fully reproducible PyTorch Lightning + Hydra codebase, ensure the research is both **implementable within ~5 days** and easy to extend.

Project Structure and Setup

To enable rapid development and clear organization, we adopt a structured project layout (inspired by Lightning + Hydra best practices [5] [6]). Below is the recommended folder structure:

```
multimodal-thesis/
├── configs/                 # Hydra configuration files
│   ├── model/               # model architecture hyperparams for each
paradigm
│   │   ├── classical_early.yaml
│   │   ├── classical_late.yaml
│   │   ├── clip.yaml
│   │   ├── vilbert.yaml
│   │   ├── mvae.yaml
│   │   └── hybrid.yaml
│   ├── data/                # dataset and dataloader configs
│   │   ├── fashion.yaml
│   │   ├── movielens.yaml
│   │   └── amazon.yaml
│   ├── trainer.yaml         # PyTorch Lightning Trainer settings (epochs,
gpus, etc.)
│   ├── train.yaml           # Default training config (specifies default
model & dataset)
│   └── eval.yaml            # Evaluation config (for retrieval, etc.)
├── data/                    # Data files or symlinks to datasets
│   ├── fashionai/...
│   ├── movielens/...
│   └── amazon/...
├── src/                     # Source code for models and training
│   ├── models/              # Model definitions for each fusion approach
│   │   ├── classical_early.py
│   │   ├── classical_late.py
│   │   ├── clip_module.py
│   │   ├── vilbert_module.py
│   │   ├── mvae_module.py
│   │   └── hybrid_module.py
│   ├── data/                # Data loading and preprocessing
│   │   ├── fashion_datamodule.py
│   │   ├── movielens_datamodule.py
│   │   └── amazon_datamodule.py
│   ├── utils/               # Utility functions (metrics, visualization,
etc.)
│   ├── train.py
# Training entry script (uses Lightning Trainer + Hydra)
│   └── evaluate.py          # Evaluation script (for cross-modal retrieval,
etc.)
├── notebooks/               # Jupyter notebooks for analysis (optional)
└── reports/                 # Saved results, model checkpoints, and figures
```

Configuration & Reproducibility: Using Hydra configs, we can easily switch between models and datasets from the command line (e.g., `python train.py model=clip data=fashion trainer.max_epochs=10`). The `train.py` will load the specified config, instantiate the Lightning `Module` and `DataModule`, and start training. This approach ensures all hyperparameters for each experiment are logged and version-controlled, crucial for reproducibility [3]. PyTorch Lightning further enforces reproducible training loops (with `pl.seed_everything(...)` for consistent runs) and reduces boilerplate, allowing us to focus on model logic [2].

Methodology and Model Paradigms

Our methodology centers on implementing multiple multimodal fusion strategies and evaluating them in a uniform framework. Each model type will be implemented as a PyTorch Lightning `LightningModule` with a consistent interface (for training and evaluation), enabling fair comparison. Below we describe each paradigm in detail, including how they will be realized in our project:

Classical Fusion Approaches (Early vs. Late Fusion)

Classical Early Fusion: Early fusion involves **concatenating raw or encoded features from each modality at the input**, then feeding them into a single model [7]. For example, for image-text data, we can take a CNN-extracted image feature vector and a text embedding (from an LSTM or BERT), concatenate them into one vector, and pass it through fully connected layers for classification. This approach learns a joint representation from the start. We will implement early fusion models using simple concatenation of modality features followed by a multilayer perceptron (MLP) classifier. Early fusion leverages interactions between modalities at the cost of a higher-dimensional input and potential overfitting if data is limited.

Classical Late Fusion: Late fusion trains **separate sub-networks for each modality and combines their outputs at a later stage (usually at the decision level)** [7]. For instance, an image CNN and a text classifier (e.g., an RNN or BERT) are trained in parallel; their penultimate features or predicted probabilities are then concatenated or averaged, and a final decision layer uses this fused information. We will implement late fusion by having modality-specific encoders produce intermediate representations which are merged (via concatenation or weighted sum) before the final output layer. This approach is modular and allows each modality to contribute independently, though it might miss some cross-modal feature interactions that early fusion captures.

Hybrid Fusion (Intermediate Fusion): We also plan to explore **hybrid fusion**, which combines elements of early and late fusion [8]. In practice, this could mean fusing modalities at multiple levels: e.g., combining some features early and others later, or performing a mid-level fusion in the network. One design is to let modalities interact at an intermediate layer (not as raw input as in early fusion, but earlier than final output). For example, we might train separate encoders, then fuse at some hidden layer and continue processing jointly. This hybrid strategy can be seen as an extension to test if fusing at a particular network depth yields better performance than strictly early or late fusion [8]. We will implement a variant if time permits (though primary classical comparisons will be early vs. late).

Note: Classical models will serve as baselines. They are relatively quick to implement (using pre-trained CNNs or transformers for feature extraction), which is advantageous given our 5-day timeline.

Transformer-Based Fusion (ViLBERT & CLIP models)

Transformer-based models use attention mechanisms to learn joint representations of multiple modalities. We focus on two prominent vision-language transformer models:

- **CLIP (Contrastive Language-Image Pretraining):** CLIP is a two-tower model with an image encoder and a text encoder trained jointly with a contrastive objective to align image and text embeddings [9] [10]. The image encoder (often a Vision Transformer or CNN) and the text encoder (Transformer like BERT) produce vectors in a shared embedding space. During training, CLIP pulls together the embeddings of matching image-text pairs and pushes apart non-matching pairs [10]. The result is a powerful multimodal representation where semantic similarity between an image and caption can be measured by cosine similarity [11] [12]. We will use a pre-trained CLIP model (via HuggingFace or OpenAI's API) to embed images and texts in our datasets, and evaluate its zero-shot or fine-tuned performance on our tasks. CLIP's ability to align modalities enables **cross-modal retrieval** naturally – we will test if, for example, a FashionAI image's embedding is closest to its correct description embedding among a gallery (measuring recall@K).

[13] [10]

```
flowchart LR
    subgraph CLIP Model
    Img[Image] -->|Vision Encoder (CNN/ViT)| iVec[Image Embedding];
    Txt[Text] -->|Text Encoder (Transformer)| tVec[Text Embedding];
    iVec & tVec -->|Contrastive Learning (train)| Shared[(Shared Embedding
Space)];
    Shared -->|Similarity Search| Retriev[Cross-modal Retrieval or
Classification];
    end
```

Diagram: CLIP architecture with dual encoders. Image and text are encoded separately into a shared latent space, trained by contrastive alignment [10]. At inference, we can perform zero-shot classification or image-text retrieval by comparing embeddings [12].

- **ViLBERT (Vision-and-Language BERT):** ViLBERT extends the BERT architecture to multimodal data by processing images and text in two streams that interact via co-attention layers [14]. An image is represented by a set of region features (e.g., from a pre-trained object detector or CNN), and text is represented by token embeddings. ViLBERT first encodes each modality independently (e.g., text through several transformer layers like standard BERT, and image regions through a preliminary CNN or embedding layer) [14]. Then, it applies **co-attentional transformer layers** where visual and textual features attend to each other, learning alignments between words and image regions [15]. This produces joint visio-linguistic representations that can be used for downstream tasks such as VQA, captioning, or retrieval. In our project, we will fine-tune ViLBERT (or a similar pretrained model like LXMERT) on image-text tasks in FashionAI (and possibly MovieLens where applicable) to compare its performance against CLIP and classical models. We will also extract its attention maps to see which words and image regions are highly attended (interpreting the model's learned alignment).

[14]

```
flowchart LR
    subgraph ViLBERT Model
    I[Image] -->|Region Feats (CNN)| Ifeat[Image Embeddings];
    T[Text] -->|BERT Encoder| Tfeat[Text Embeddings];
    Ifeat -->|Co-attention| Joint[Joint Visio-Linguistic Rep];
    Tfeat -->|Co-attention| Joint;
    Joint -->|Task-specific Head| Output[Prediction];
    end
```

Diagram: ViLBERT architecture (simplified). Image features and text tokens are encoded separately, then fused by co-attention transformer layers [16]*, producing a joint representation used for tasks (e.g., answer prediction, retrieval ranking).*

Note: Both CLIP and ViLBERT represent **state-of-the-art transformer-based fusion**: CLIP excels at **open-vocabulary recognition and retrieval** [17], whereas ViLBERT excels at **task-specific fine-tuning** (e.g., question answering with aligned image regions). By including both, we compare contrastive vs. co-attentional fusion mechanisms. We will leverage pre-trained weights to save time (CLIP's pretrained embeddings; ViLBERT's pretrained on Conceptual Captions [18]) and fine-tune on our datasets as needed. This reuse of pretrained models aligns with our 5-day implementation window, avoiding training huge models from scratch.

Generative Fusion: Multimodal Variational Autoencoder (M-VAE)

Multimodal VAE: Variational Autoencoders provide a generative approach to learn a **shared latent space** for multiple modalities [1]. A multimodal VAE has modality-specific encoders that map each modality to a common latent vector *z*, and decoders that reconstruct each modality from *z*. The VAE objective ($L = L_{recon} + L_{KL}$) encourages *z* to capture the joint distribution of the modalities. Such models naturally handle missing modality inputs (by inference over *z*) and can **separate shared factors from modality-specific noise** [19]. In our project, we will implement an M-VAE for each dataset: for instance, on FashionAI (image+text), the encoder will take both an image and its text description and produce a latent code *z* (we may concatenate image and text features then pass through an encoder network to get $q(z|image,text)$). The decoders will then try to reconstruct the image (or some image features) and the text (or text features) from *z*. Training such a model forces *z* to represent information sufficient for both modalities, ideally capturing their semantic intersection.

[1]

```
flowchart LR
    subgraph Multimodal VAE
    Img1[Modality A] --> E1[Encoder_A];
    Txt1[Modality B] --> E2[Encoder_B];
    E1 --> || Z((Latent z)) ||;
    E2 --> || Z ||;
    Z --> D1[Decoder_A] --> Img_rec[Recon A];
    Z --> D2[Decoder_B] --> Txt_rec[Recon B];
    end
```

Diagram: Multimodal VAE for two modalities. Encoders E_A, E_B encode each modality into a shared latent variable z, from which decoders attempt to reconstruct the original inputs. The VAE's latent space learns a joint representation of both modalities.

We will likely use a **simplified VAE** implementation due to time constraints – for example, using lower-dimensional latent (e.g., $|z|=50$) and perhaps reconstructing *features* rather than raw data (to avoid heavy image generation). For instance, the image decoder could reconstruct high-level image features or class logits instead of pixels, and the text decoder could reconstruct bag-of-words or an embedding rather than full sentences. The focus is to learn a *common latent space* that we can evaluate for semantic alignment (by seeing if *z* captures aligned concepts: e.g., cluster similar items regardless of modality) and for disentangled factors (testing if some dimensions of *z* correspond to meaningful attributes).

Hybrid Fusion Model (M-VAE + Classifier)

The **Hybrid model** combines the unsupervised strength of a VAE with a supervised classifier, by feeding the VAE's latent representation into a prediction network. This is inspired by Khattar et al.'s MVAE for fake news, where a bimodal VAE's latent was fed to a classifier [1]. In our design, we will train the multimodal VAE simultaneously with a classification objective: *z* goes through an additional feed-forward network to predict the target (e.g., product category, user rating) alongside the VAE's reconstruction of inputs. The loss is a weighted sum of VAE reconstruction loss and classification loss. This hybrid training can improve *z*'s usefulness for the task while retaining generative alignment capabilities [4].

[1] [4]

```
flowchart LR
    subgraph Hybrid VAE+Classifier
    Img2[Modality A] --> EncA;
    Txt2[Modality B] --> EncB;
    EncA --> Z2((Latent z));
    EncB --> Z2;
    Z2 --> DecA --> Img_rec2;
    Z2 --> DecB --> Txt_rec2;
    Z2 --> Cls[Classifier] --> Yhat[Predicted Output];
    end
```

Diagram: Hybrid Fusion model. The latent z from the VAE (shared by both modalities) is fed into a classifier head to predict task labels, while simultaneously being used to reconstruct inputs. This joint training encourages z to be both generatively meaningful and predictive [1].

We will implement the hybrid model as an extension of the M-VAE LightningModule: adding an output layer for classification and combining losses. For example, on FashionAI we might predict clothing category from *z*; on MovieLens, predict a user's rating or a movie genre; on Amazon, predict the review sentiment or rating. The novelty here is leveraging the **generative latent space for supervised learning** – we expect this could improve performance if *z* captures complementary multimodal features, and also enforce that *z* is interpretable (since it must decode to inputs and correlate with outputs).

Interpretability in VAE/Hybrid: We will analyze the learned latent space by plotting *z* distributions, performing **latent traversals** (vary one dimension of *z* to see effect on decoded outputs), and checking if any dimension correlates with known factors (e.g., perhaps one dimension correlates with sentiment or product type). The literature suggests multimodal VAEs can separate shared vs. private factors [19]; our hybrid model's supervised component might further align some latent dimensions with label information (which we can verify by examining latent activations).

Datasets and Experimental Plan

We utilize three datasets, each adding complexity in modalities:

- **FashionAI (image + text):** A fashion dataset containing clothing images paired with textual information (e.g., item descriptions, attributes). This dataset provides a **bimodal image-text scenario** similar to classical vision-language tasks. We will use FashionAI in **Phase 1** to conduct foundational experiments: testing image-text multimodal models (CLIP, ViLBERT, etc.) on tasks like *image-text matching* or *attribute classification*. For example, given an image of a dress and a description, models should align them in embedding space (for retrieval) and possibly classify the item's attributes or category. This dataset is ideal for benchmarking our pipelines because it's compact and well-defined as an image-text pair problem.

- **MovieLens Enriched (text + image + graph + numeric):** We extend the MovieLens 1M/20M dataset (user-movie ratings) with additional content modalities:

- *Text:* Movie synopses or tag lines (provides semantic content about the movie).
- *Image:* Movie posters (visual content).
- *Graph:* A user-item interaction graph or a knowledge graph of movie connections (e.g., genre or actor relationships). For simplicity, we may represent the user-movie interactions as a bipartite graph or leverage user ID and movie ID embeddings (this implicitly encodes graph info of the ratings matrix).
- *Numeric:* Structured data like average rating, release year, or movie metadata (budget, etc.) – or simply the rating itself as a feature for training a predictor (if doing a self-supervised task).

MovieLens in **Phase 2** allows a **multi-modal (tri- or quad-modal) fusion** evaluation. Our primary task here could be rating prediction (a regression or classification if we bucket ratings), or a recommendation ranking task. We will compare: - Classical fusion: e.g., early fusion concatenating all available features (text embedding + image features + user/movie IDs or graph embeddings + numeric features) into one vector to predict rating. Late fusion: separate sub-networks for text, image, etc., merging at the end. - Multimodal VAE: an encoder that integrates all modalities into *z*, and decoders that could reconstruct each (this is challenging with many modalities, but we can attempt a scaled-down version focusing on text-image-user features). - Hybrid: VAE latent to predict rating. - Transformer-based: CLIP/ViLBERT are inherently bimodal (image-text), so they cannot directly fuse 4 modalities. However, we can still utilize CLIP embeddings for image+text (poster + synopsis) as part of a larger model (e.g., use CLIP to get a joint image-text embedding, then combine with numeric and graph features in a late fusion manner). Similarly, we might use a graph neural network or matrix factorization for the user-movie graph part and combine it. Due to time limits, a full GNN is likely out of scope; we may instead use precomputed user and item latent factors (from a simple recommender model) as the “graph” features.

MovieLens experiments will demonstrate how models scale to more than two modalities. We anticipate classical early fusion might struggle if one modality dominates, whereas a VAE could help learn a balanced latent representation. Evaluation will focus on rating prediction error (RMSE) and possibly a top-K recommendation metric (NDCG or Recall@K) if we frame it as recommendation.

- **Amazon Reviews (text + metadata):** A large collection of product reviews with review text (and possibly product images or category info; here we assume *text plus tabular metadata* since Path A says text + metadata). The metadata could include things like star rating, product category, or review timestamp. This dataset gives a **mixed-modality** scenario where one modality is natural language and the other is structured data. Our task can be sentiment classification or star rating prediction from the review text + metadata. For example, predict if a review is positive or negative using the review text plus clues like product category or verified purchase (metadata). Another angle: predict the numeric star rating (1-5) from the review text (a regression), where metadata like product type might help the model calibrate expectations (some categories might have generally higher ratings, etc.).

In **Phase 2**, Amazon allows testing models on *text* + *numeric* data: - Classical early fusion: concatenate text embedding (from BERT or an LSTM) with a vector of metadata features (e.g., one-hot encoded categories, normalized numeric fields) and feed to an MLP. - Late fusion: e.g., a text sentiment model and a separate metadata model whose outputs are merged. - Transformer-based: Since there’s no image, CLIP/ViLBERT don’t apply. However, we could use a Transformer for text (like BERT) and perhaps treat metadata as an additional token sequence or as extra inputs to a classifier layer. If time permits, we might try a simple approach of encoding metadata as text (e.g. appending "Rating: 5 stars" to the review text) to see if a language model can ingest it. But likely, classical methods suffice here. - M-VAE: treat text and metadata as two “modalities” – encode text via an LSTM/BERT encoder, metadata via a small encoder (e.g., an MLP for the numerical features), combine into *z*, and reconstruct both (maybe reconstruct text features and exact metadata values). - Hybrid: use *z* to predict the sentiment label or star rating, while also reconstructing inputs.

This dataset will highlight how our fusion methods handle modalities of very different nature (free-form text vs. structured data). Performance will be measured by classification accuracy or regression error on ratings, and we will particularly see if the VAE/hybrid approach helps in capturing relationships like certain words corresponding to certain star ratings.

Phased Execution Plan: We will proceed in two phases to manage scope and time:

- **Phase 1 (Days 1-2):** Focus on **FashionAI (image-text)** experiments. Implement and debug the core pipeline with simpler bimodal data. We will run:
- CLIP and ViLBERT on image-text (possibly using pre-trained models and evaluating zero-shot or fine-tuning lightly on a supervised task like attribute classification or matching).
- Classical early/late fusion on image-text (e.g., CNN + BERT early concat vs. separate then late merge).
- M-VAE on image-text (train a VAE to reconstruct image features and text).
- Hybrid VAE+classifier on image-text (train VAE that also classifies an attribute).

We will evaluate retrieval (matching images with correct text descriptions) to quantify **semantic alignment**. For instance, use the image encoder and text encoder parts of each model to embed the validation set and compute if the closest text embedding to an image embedding is the true match (measuring Recall@1,5,10). We will also compute standard classification metrics if a label is available (e.g., clothing type prediction accuracy). During this phase, the emphasis is on ensuring each model code works and gathering preliminary results.

- **Phase 2 (Days 3-5):** Integrate **MovieLens and Amazon** datasets for extended multimodal experiments:
- For MovieLens: prepare data loaders that provide all modalities for a given movie (and user if needed). Train classical fusion models to predict ratings. Possibly fine-tune M-VAE and hybrid on the combined modality input (this is ambitious, so if needed we could simplify by focusing on a subset, such as only movie content modalities and ignoring user for the generative part, using user as just another feature in classifier). Evaluate RMSE on rating prediction. Also, evaluate an alignment proxy: e.g., given a movie poster, retrieve the correct plot description from a pool, to test image-text alignment even in this multi-modal context.
- For Amazon: train fusion models to predict sentiment from text+metadata. Evaluate classification accuracy (or MAE if predicting star rating). Because cross-modal retrieval is not applicable (text vs. metadata retrieval is not intuitive), focus on latent space analysis instead: e.g., see if the VAE's latent separates positive vs negative reviews (visualize with t-SNE).
- Apply the **Hybrid model** on both: see if adding the classifier loss helps the VAE latent predict better. For example, compare the VAE-only vs. Hybrid's classification performance.

Throughout Phase 2, we will also incorporate **interpretability** analysis for the complex models: - Use **Grad-CAM** on any CNN image encoders (e.g., the CNN in early fusion or the one inside ViLBERT) to highlight image regions important for the prediction [20]. For instance, in MovieLens, Grad-CAM on the poster image might show the model focusing on a certain actor's face or a genre-indicative scene when predicting a high rating. - Extract **attention maps** from transformers: e.g., in ViLBERT, look at the co-attention matrix for a sample to see which words attend strongly to which image regions (this can be visualized by overlaying attention on the image as boxes or by highlighting words). In text-only cases (Amazon), if using BERT, we can visualize self-attention to see if words like "excellent" get high attention weights. - **Latent space analysis:** For VAE-based models, perform qualitative checks on latent disentanglement. For example, in FashionAI, vary one latent dimension and decode the image to see if it changes color or style while text changes accordingly (this might be hard without a sophisticated decoder, but we can instead check if latent dimensions correlate with known attributes: e.g., one dimension might correlate with "formal vs casual" if such attribute is in text). In MovieLens, see if latent *z* clusters movies by genre or user preference. In Amazon, see if *z* clusters by sentiment.

We will document observations from these analyses to provide insight into **why** certain models perform better and how they represent multimodal data.

Execution Workflow and Modular Training Design

To ensure the project is completed in ~5 days, a clear execution workflow is essential. Below is a step-by-step plan, emphasizing modularity and re-use:

1. **Environment Setup:** Use a consistent environment (e.g., `conda` or `venv`) with Python 3.x, and install PyTorch, PyTorch Lightning, Hydra, and any necessary libraries (HuggingFace Transformers for CLIP/ViLBERT, Torchvision for CNNs, etc.). Verify GPU availability. Initialize a git repository for version control of code and configs (optional but recommended for tracking changes).

2. **Data Preparation:** Download/collect the datasets:

3. FashionAI: Ensure we have image files and accompanying texts/labels. Write a LightningDataModule (`fashion_datamodule.py`) to handle loading images and texts, applying transforms (e.g., image resizing, tokenizing text with BERT tokenizer), and batching. Use Hydra config to set paths and batch size.
4. MovieLens: Download ratings and movie metadata. Use external sources or provided data for posters and plots if available (we might scrape IMDB or use a precompiled dataset that includes them [21]). Preprocess: e.g., encode user IDs and item IDs (we can embed these or one-hot them), tokenize plot text, load images. The `movielens_datamodule.py` will collate these features. Because of complexity, test this pipeline on a small subset first.
5. Amazon Reviews: Possibly use a subset (to manage size). Preprocess text (tokenize) and normalize metadata (e.g., rating as float, categories as one-hots). DataModule will yield text and metadata tensors, and label (if supervised like sentiment).

6. **Tip:** Use Hydra configs to easily switch between small/large dataset versions (for quick debugging use a tiny sample, for actual run use full data).

7. **Model Implementation:** For each paradigm, implement a LightningModule in `src/models/`:

8. *Classical Early Fusion Module:* Takes multi-modal inputs, immediately concatenates features, then forward through a classifier (e.g., an MLP or a simple linear if using pre-extracted features). Define `training_step` with loss (cross-entropy or MSE).
9. *Classical Late Fusion Module:* Has separate sub-networks (e.g., define two PyTorch modules within it, one for image, one for text). In `forward`, process each modality, then fuse outputs (concatenate or add) and apply final layer. Loss similar to above.
10. *CLIP Module:* We likely won't train CLIP from scratch. Instead, we create a module that loads a pre-trained CLIP model (using `CLIPModel.from_pretrained('openai/clip-vit-base-patch32')` for example) [22] [23]. In `forward`, we can either output the similarity of image-text pairs (if doing retrieval or matching task) or attach a small classifier head for a specific label (fine-tuning scenario). Freeze or fine-tune weights depending on results (freezing initially to save time).
11. *ViLBERT Module:* If a pre-trained ViLBERT (or similar model like LXMERT) is accessible via HuggingFace, load it and fine-tune on our tasks. If not, as a fallback, we could simulate a small version: e.g., use a pretrained BERT for text and a ResNet for image, then implement a simple cross-attention layer ourselves. Given time constraints, we lean on existing implementations

(HuggingFace's `VilBERTForQuestionAnswering` or `LxmertForPreTraining` can be repurposed). The LightningModule will manage inputs to these models and compute loss.

12. *M-VAE Module:* Implement encoder and decoder networks. For simplicity, use fully connected layers on top of precomputed features: - The **encoder** could take concatenated modality features (e.g., image feature vector + text feature vector) and output μ and σ for Gaussian latent *z*. Alternatively, use modality-specific encoders that output their own μ, σ and combine them (e.g., via average or concatenation) into one joint latent distribution [24] (due to time, concatenation then a small network to get final μ, σ is acceptable). - The **decoder** for each modality takes a sample of *z* and tries to reconstruct the input features. E.g., decode *z* to an image feature vector (matching the CNN feature extracted from the image) and to a text feature (matching the BERT [CLS] embedding for the text). We'll use mean squared error or cross-entropy (if reconstructing discrete features) for reconstruction loss. - Training will involve KLD loss on *z* (to keep distribution close to standard normal) plus reconstruction losses. We might weight losses for each modality appropriately (especially if one modality's scale is larger).
13. *Hybrid Module:* Extend M-VAE Module: add a classifier output. For instance, after obtaining *z*, pass it through an MLP to predict the label (e.g., fake vs real in Khattar's case, but here whatever our task label is). Use a loss like cross-entropy for classification and add it to the VAE loss (with some weighting factor λ to balance supervised vs. unsupervised objectives). In training, log both reconstruction and classification losses for monitoring.

Each LightningModule will implement `configure_optimizers` (likely using Adam), and if needed `validation_step` for logging validation metrics.

1. **Training & Hyperparameters:** Use PyTorch Lightning Trainer to manage training loops (with early stopping or model checkpointing callbacks as needed). We will likely train each model for a modest number of epochs (due to time, e.g., 5-10 epochs or until convergence if earlier). Use the Hydra configs to set different hyperparameters:
2. E.g., learning rates might differ: the VAE might need a slightly lower LR for stability, CLIP fine-tuning might require a very low LR.
3. Batch sizes depending on dataset (FashionAI images might allow batch 32, but for MovieLens, if we load heavy modalities, maybe batch 16).
4. We will keep runs short and possibly use mixed precision (fp16) to speed up if supported.

5. Each night, longer training jobs (if needed for VAE) can be run, but priority is to get useful results quickly (Lightning's built-in progress and checkpointing helps recover if needed).

6. **Evaluation Procedure:** After training each model (Lightning makes it easy to save checkpoints), we perform evaluations:

7. **Performance Metrics:** For classification tasks, compute accuracy, F1-score, etc. For regression (ratings), compute RMSE or MAE. Lightning can log these during validation, but we will also run the `evaluate.py` script to aggregate final metrics for each model on each dataset for comparison.
8. **Cross-modal Retrieval:** Write a function to perform retrieval: e.g., embed all images and texts with the model's encoders, then for each image find the closest text (using cosine similarity in CLIP space or Euclidean in latent space). Compute metrics like Recall@K, Mean Reciprocal Rank. We will do this for FashionAI (and possibly for movie posters vs plots). This will demonstrate the **semantic alignment** ability of each model's latent representation [25].

9. **Latent Space Visualization:** Use tools like PCA or t-SNE on the latent vectors from M-VAE/Hybrid for a sample of data. Plot them with color-coding by class label (or some attribute) to see clustering.
10. **Grad-CAM and Attention Maps:** For some sample inputs, generate Grad-CAM heatmaps [20] for image models. This involves taking a trained model (e.g., the CNN part of early fusion or the ViLBERT's visual encoder) and using a library like `pytorch-grad-cam` to produce a heatmap highlighting important pixels for the prediction [20]. Save these heatmaps in `reports/` folder. Similarly, extract attention weights from the transformer models for a couple of examples and visualize (for example, show which part of an image was highly attended by a certain word in ViLBERT).
11. **Interpretation of Results:** Aggregate findings such as:
    - Which model had highest accuracy on each task? (performance comparison)
    - Are there cases where one model strongly outperforms others? (e.g., maybe CLIP excels in retrieval, Hybrid VAE might do nearly as well while also being generative.)
    - Do the cross-modal retrieval results correlate with supervised performance? (This can indicate if semantic alignment goes hand-in-hand with task performance or not.)
    - What do the interpretability analyses show? (e.g., perhaps classical models focus on obvious features, while ViLBERT's attention reveals nuanced associations between words and visual elements; the VAE's latent might capture some high-level factor like "is clothing formal or casual".)

Throughout execution, Hydra config makes it trivial to reproduce an experiment or tweak parameters. For example, if initial runs show the hybrid model underperforming due to imbalance between reconstruction and classification loss, we can adjust the weight in config and re-run that component. Similarly, if ViLBERT is too slow, we might skip it or run fewer epochs.

Modular Training Script (Hydra + Lightning): The `train.py` script will tie everything together. Pseudo-code:

```
# train.py
import hydra
from omegaconf import DictConfig
from pytorch_lightning import Trainer, seed_everything
from src import models, data  # assume __init__.py in those dirs for easy import

@hydra.main(config_path="configs/", config_name="train.yaml")
def main(cfg: DictConfig):
    seed_everything(cfg.seed)
    # Instantiate DataModule and Model based on config
    dm = data.DataModuleRegistry.get(cfg.data.name)(cfg.data)        # e.g.,
FashionDataModule(cfg)
    model = models.ModelRegistry.get(cfg.model.name)(cfg.model, cfg) # e.g.,
CLIPModule(cfg)
    trainer = Trainer(**cfg.trainer)  # unpack trainer configs (gpus, epochs,
etc.)
    trainer.fit(model, datamodule=dm)
    trainer.validate(model, datamodule=dm)  # evaluate on val set
```

```python
if __name__ == "__main__":
    main()
```

This design uses registry or conditional logic to select the correct classes based on `cfg.model.name` (one of: "classical_early", "clip", "mvae", etc.) and similar for data. This way, **one script trains any model/data combo** by just changing the config. We will also have an `evaluate.py` to possibly load a saved model checkpoint and run full test evaluations (including custom retrieval evaluations, which might not be part of Lightning's built-in loop).

Thesis Writing Outline

Finally, to organize the thesis document, here is a proposed outline for the **Methodology** and **Results** chapters, ensuring all important components are covered:

Methodology Chapter Outline

- **Introduction to Methodology:** Recap of research objectives and approach. (Reiterate the paradigms being compared and why.)
- **Datasets and Preprocessing:**
- Describe each dataset (FashionAI, MovieLens, Amazon) – what modalities and tasks, why they were chosen.
- Preprocessing steps for each (image resizing, text tokenization, feature normalization, etc.).
- Summary table of dataset stats (e.g., number of instances, modality types, example features).
- **Model Architectures:**
- *Classical Fusion Models:* Definitions of early, late, hybrid fusion with diagrams. Include model architecture figure of the implemented version (e.g., how image and text features are concatenated) [7].
- *Transformer-Based Models:* Description of CLIP and ViLBERT architectures [9] [14]. Explain any modifications for our use (e.g., using pre-trained vs fine-tuned, how ViLBERT was adapted for our data). Include an architecture diagram highlighting dual encoders and co-attention.
- *Multimodal VAE:* Explain VAE design (encoder/decoder structure) and how it handles multimodal input [1]. Possibly include the VAE graphical model or network diagram.
- *Hybrid VAE-Classifier:* Detail the combined loss function and architecture [4]. Emphasize why this is included – hypothesize benefits.
- *Summary of Models:* A comparative table of model characteristics (e.g., number of parameters, whether pre-trained, how modalities are fused).
- **Training Setup:**
- Framework and tools: Justify use of PyTorch Lightning and Hydra for training (reproducibility, modularity) [2] [3].
- Hyperparameters and optimization: list learning rates, batch sizes, training duration for each model. Mention any tuning done.
- Hardware used: e.g., "Trained on single NVIDIA V100 GPU, training time ~2 hours per model" (for context).
- Any tricks for stabilization: e.g., for VAE (KL annealing or beta-VAE if used), for hybrid (loss weighting).
- **Evaluation Methods:**
- Performance metrics: define accuracy, F1, RMSE, etc., for each task.

- Cross-modal retrieval evaluation: describe the procedure to measure retrieval alignment (e.g., use cosine similarity in shared space; compute recall@K) [25] .
- Interpretability analysis: describe how attention maps and Grad-CAM are used to probe models [20] . Also describe latent space visualization and disentanglement evaluation (possibly mention any metric like silhouette score for clusters or qualitative inspection).

- Explain that these evaluations address not just "how well" but also "how" the models learn multimodal data.

- **Implementation Details (if needed):**

- Outline any important implementation decisions: e.g., using pre-trained weights, any limitations (like not all models applied to all datasets – clarify which models were evaluated on which dataset and why).
- Perhaps include code snippets or pseudo-code (LightningModule structure) in an appendix if appropriate, but main text can summarize.

Results Chapter Outline

- **Quantitative Performance Comparison:**
- *FashionAI Results:* Table or charts comparing all models on FashionAI (accuracy in matching or classification, retrieval recall, etc.). Discuss which models performed best and analyze why (e.g., "CLIP achieved highest retrieval accuracy [17] , likely due to its rich pre-training, while the hybrid model was close behind, suggesting the generative approach effectively learned the image-text alignment.").
- *MovieLens Results:* Present results on rating prediction (e.g., a table of RMSE for each model). Also include any retrieval or auxiliary evaluation (maybe image-text retrieval on posters vs plots). Discuss findings (e.g., perhaps the hybrid VAE handled the heterogeneous data well, or classical early fusion outperformed others due to linear nature of the task).
- *Amazon Results:* Show accuracy or error for text+metadata sentiment prediction. Possibly the difference between using metadata or not (to show fusion value). Discuss which model did best (maybe classical early fusion with BERT did very well since text is dominant, etc.).
- *Across-Task Comparison:* A brief discussion comparing model trends across datasets. For example, note if a model that's best for image-text might not be best for text+metadata, etc., highlighting the importance of choosing the right approach per data type.
- **Semantic Alignment & Retrieval Analysis:**
- Present results of cross-modal retrieval experiments (perhaps as a small table of Recall@1 for image→text and text→image retrieval for applicable datasets and models). Discuss how well each model's latent space aligned the modalities. (e.g., "CLIP and ViLBERT nearly perfect on FashionAI retrieval, whereas early fusion (with no explicit alignment training) performed worse, confirming the importance of contrastive learning for alignment [26] .").
- If available, include a figure plotting the joint embeddings (e.g., 2D t-SNE where image and text points are shown). Point out if same-item image/text pairs are close in that space for each model.
- Discuss semantic alignment in MovieLens if tested (perhaps image vs text vs others – though tricky, maybe mention if VAE latent brought text and image of a movie closer than they were in raw feature space).
- **Interpretability and Latent Space Analysis:**

- Show example **attention maps** or **Grad-CAM images** (as figures) for a few cases:
    - E.g., an image with Grad-CAM highlighting the region the model focused on (with a caption like "Grad-CAM for classical model on a dress image shows focus on the lace pattern when predicting 'lace dress' class" [20]).
    - An example of ViLBERT co-attention: perhaps a table of word-region attention weights or an image with bounding boxes highlighted for a given word (e.g., word "guitar" attends to the guitar in the image).
    - If possible, a visualization of **hybrid VAE's latent traversal**: e.g., decode two images by varying one latent factor to demonstrate a semantic change.
- Discuss these findings: e.g., "The ViLBERT attention visualization indicates the model correctly learned to ground certain words to image regions (the word *"dog"* attends to the dog's face in the image, demonstrating interpretability of the co-attention mechanism). Classical early fusion, being a black-box MLP, is harder to interpret, but Grad-CAM on its CNN part shows it attends to relevant regions as well [20] . The VAE's latent factors were explored: we found one latent dimension correlated with review sentiment (high values for positive words), hinting at disentanglement."
- If any quantitative measure of interpretability was done (e.g., attention flow or completeness), report that too, though likely this will be mainly qualitative.
- **Error Analysis:**
- Analyze where models made mistakes. For instance, were there certain types of images or text where one model failed? Did the VAE reconstructions reveal any mode collapse or missing details? This can be a brief subsection giving a couple of illustrative failure cases and how interpretability helped diagnose them.
- **Discussion:**
- Summarize which approach is most suitable under what conditions. For example:
    - "Transformer models like CLIP/ViLBERT excel in image-text tasks given large-scale pre-training [17] , but their complexity and need for dual inputs make them less flexible for more than two modalities."
    - "Classical early/late fusion is straightforward and performed competitively on text+metadata where a simpler model sufficed, but it underperformed on tasks requiring fine cross-modal alignment (FashionAI retrieval) because it doesn't explicitly align modalities during training."
    - "The Multimodal VAE achieved decent performance and provides generative capabilities; the hybrid VAE notably improved supervised performance over the pure VAE, confirming that the inclusion of label supervision helps the latent space become more task-informative [1] . However, the VAE-based methods were somewhat more complex to train (e.g., needing tuning of the loss balance) and slower per epoch due to decoding overhead."
- Mention limitations: e.g., "Due to time constraints, ViLBERT was only fine-tuned briefly – given more time it might have improved. The Amazon dataset's sheer size meant we used a subset, which could affect generalizability. We also simplified the graph modality in MovieLens (did not implement a full graph neural network). These could be addressed in future work."
- End with a forward-looking statement: how these findings could guide choosing multimodal models or inspire new hybrid approaches (e.g., combining contrastive and generative training, or applying these insights to other domains like audio-visual, etc.).

This comprehensive outline ensures the thesis covers the implementation details and the comparative analysis in a clear, logical manner. Each chapter section maps to the objectives: methodology explains *how* and *what* was done, while results explain *what was found* and *why it matters*. All guidance here is geared toward clarity and reproducibility – from well-structured code to interpretable results – aligning with the 5-day implementation plan and the overarching research goals.

[1] [4] MVAE: Multimodal Variational Autoencoder for Fake News Detection | Request PDF
https://www.researchgate.net/publication/333076175_MVAE_Multimodal_Variational_Autoencoder_for_Fake_News_Detection

[2] [3] [5] [6] GitHub - ashleve/lightning-hydra-template: PyTorch Lightning + Hydra. A very user-friendly template for ML experimentation. ⚡⚡
https://github.com/ashleve/lightning-hydra-template

[7] [8] Multimodal Models and Computer Vision: A Deep Dive
https://blog.roboflow.com/multimodal-models/

[9] [10] [11] [12] [13] [17] [22] [23] What is CLIP: Contrastive Language-Image Pre-Training?
https://www.marqo.ai/course/introduction-to-clip-and-multimodal-models

[14] [15] [16] [18] Paper Summary — ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks | by Aditya Chinchure | Technonerds | Medium
https://medium.com/technonerds/paper-summary-vilbert-pretraining-task-agnostic-visiolinguistic-representations-for-44b0eeea1257

[19] [24] Disentangling shared and private latent factors in multimodal Variational Autoencoders
https://arxiv.org/html/2403.06338v1

[20] A Guide to Grad-CAM in Deep Learning - Analytics Vidhya
https://www.analyticsvidhya.com/blog/2023/12/grad-cam-in-deep-learning/

[21] MovieLens Data Enrichment — Merlin HugeCTR documentation
https://nvidia-merlin.github.io/HugeCTR/main/notebooks/multi-modal-data/02-Data-Enrichment.html

[25] Cross-Modal Retrieval | Papers With Code
https://paperswithcode.com/task/cross-modal-retrieval

[26] A self-supervised framework for cross-modal search in ... - Nature
https://www.nature.com/articles/s41598-024-60256-7