GILBERTINES READING GROUP

AI Research, Hands-on

Skills that still matter in the LLM era.

✓ **Practical skills**

✗ No big picture: trends & predictions, mindset, shiny new tools

Xu Dong

University of Surrey · Gilbertines

28 / 10 / 2026

WHAT THE RIDE TAUGHT ME

Six things I wish I'd known in year one.

01

Rejection is feedback, not a verdict.

Every paper I have was rejected or almost rejected first.

02

The flat months are part of the curve.

Six months of nothing is normal. Breakthroughs tend to arrive late.

03

Know when to quit.

Check feasibility, then let go. A miracle might happen, but the odds aren't worth the months.

04

Don't bet everything on one venue.

Parallel projects and backup venues keep the pipeline moving.

05

Keep reading, learning, thinking.

The field changes every month. Read every week, and think before you prompt.

06

Use your tools well.

W&B to track runs, coding agents for the boilerplate, Zotero and AI to organise papers.

Undergrad CS: learn all of this.

61 topics · 3 years · 0 ChatGPT

Software Eng. Agile · Scrum · Git · UML · Design Patterns · Requirements · Code Review · Unit Testing

Programming Java · Python · C++ · Ruby · HTML/CSS · JavaScript · SQL · Haskell

Syntax Pointers · if / while / for · class · OOP · Exceptions

Algorithms Dynamic Programming · Greedy · Backtracking · Recursion · Big-O · Divide & Conquer · Binary Search · Sorting · BFS/DFS · Dijkstra · MST

Data Structures Array · Stack · Queue · Linked List · Hash Table · Set · Tree · BST · Heap · Graph

Computer Systems Binary & Hex · Two's Complement · Floating Point · CPU & Cache · Memory · Processes & Threads · File Systems · TCP/IP · Databases

Theory & Maths Discrete Maths · Logic · Set Theory · Automata · Computability · Linear Algebra · Probability · Statistics · Information Theory · Convex Optimization

LLM era: what you still need.

20 kept · 41 crossed out, ask the model

Software Eng. ~~Agile~~ ~~Scrum~~ **Git** ~~UML~~ ~~Design Patterns~~ ~~Requirements~~ **Code Review** **Unit Testing**

Programming ~~Java~~ **Python** ~~C++~~ ~~Ruby~~ ~~HTML/CSS~~ ~~JavaScript~~ ~~SQL~~ ~~Haskell~~

Syntax ~~Pointers~~ ~~if / while / for~~ ~~class~~ **OOP** ~~Exceptions~~

Algorithms **Dynamic Programming** ~~Greedy~~ ~~Backtracking~~ **Recursion** **Big-O** ~~Divide & Conquer~~ ~~Binary Search~~ ~~Sorting~~ ~~BFS/DFS~~ ~~Dijkstra~~ ~~MST~~

Data Structures **Array** ~~Stack~~ ~~Queue~~ ~~Linked List~~ **Hash Table** ~~Set~~ **Tree** ~~BST~~ ~~Heap~~ **Graph**

Computer Systems ~~Binary & Hex~~ ~~Two's Complement~~ **Floating Point** **CPU & Cache** **Memory** ~~Processes & Threads~~ ~~File Systems~~ ~~TCP/IP~~ ~~Databases~~

Theory & Maths ~~Discrete Maths~~ **Logic** ~~Set Theory~~ ~~Automata~~ ~~Computability~~ **Linear Algebra** **Probability** **Statistics** **Information Theory** ~~Convex Optimization~~

Then you enter AI. Learn this too.

89 more topics · and counting

Tasks Classification · Detection · Segmentation · Pose Estimation · Tracking · Image-to-Text · VQA · Action Recognition · TAL · Dense Captioning · Action Anticipation · AQA · Video QA · Retrieval

Engineering PyTorch · HuggingFace · Mixed Precision · DDP/FSDP · DeepSpeed · vLLM · Quantization · W&B

LLM Era Pre-training · SFT · LoRA · RLHF/DPO · Tokenization · CLIP · VLMs · Prompting & CoT · Evaluation · Scaling Laws · RAG · Agents · MoE

Generative Models Autoregressive · Diffusion · VAE · GAN · Autoencoder · Flows

Architectures CNN · AlexNet · VGG · ResNet · RNN · LSTM · Attention · Transformer · ViT · GCN · U-Net

Neural Nets Perceptron · MLP · Activations · Backprop · SGD · Adam · LR Schedule · BatchNorm · LayerNorm · Residuals · Dropout · Initialization

Paradigms Supervised · Unsupervised · Self-supervised · Semi-supervised · Weakly-supervised · Transfer Learning · Zero-shot · One-shot · Few-shot

Classic ML Loss Functions · Gradient Descent · Overfitting · Bias-Variance · Regularization · Cross-Validation · Metrics · Linear Regression · Logistic Regression · PCA · k-means · MLE/MAP · PAC/VC Theory · SVM · Trees & Forests

Inside AI, the foundations still matter.

50 kept · 24 ask the model

Engineering PyTorch · HuggingFace · Mixed Precision · ~~DDP/FSDP~~ · ~~DeepSpeed~~ · ~~vLLM~~ · ~~Quantization~~ · ~~W&B~~

LLM Era Pre-training · SFT · LoRA · RLHF/DPO · Tokenization · CLIP · VLMs · Prompting & CoT · Evaluation · ~~Scaling Laws~~ · ~~RAG~~ · ~~Agents~~ · ~~MoE~~

Generative Models Autoregressive · Diffusion · ~~VAE~~ · ~~GAN~~ · ~~Autoencoder~~ · ~~Flows~~

Architectures CNN · ~~AlexNet~~ · ~~VGG~~ · ResNet · ~~RNN~~ · ~~LSTM~~ · Attention · Transformer · ViT · GCN · ~~U-Net~~

Neural Nets Perceptron · MLP · Activations · Backprop · SGD · Adam · LR Schedule · BatchNorm · LayerNorm · Residuals · ~~Dropout~~ · ~~Initialization~~

Paradigms Supervised · Unsupervised · Self-supervised · Semi-supervised · Weakly-supervised · Transfer Learning · Zero-shot · One-shot · Few-shot

Classic ML Loss Functions · Gradient Descent · Overfitting · Bias-Variance · Regularization · Cross-Validation · Metrics · Linear Regression · Logistic Regression · PCA · k-means · ~~MLE/MAP~~ · ~~PAC/VC Theory~~ · ~~SVM~~ · ~~Trees & Forests~~

AI is strong. You still need a knowledge pool.

AI today can...

Write and debug code in minutes.

Reproduce a paper from its PDF.

Summarise a new field in an afternoon.

Run experiments end to end with coding agents.

Draft, polish and critique your writing.

Explain any concept, at any level.

But it only answers the questions you know to ask.

Your knowledge pool

Maths | Architectures | Loss functions | Training tricks | Failure patterns | Classic papers | Your domain

Propose ideas

Spot that an old trick fits a new problem.

"Long clips fail. A hierarchical or memory model?"

Design methods

Know the options and their trade-offs.

"Fuse pose and video: concat, gating or cross-attention?"

Analyse problems

Recognise the pattern behind a symptom.

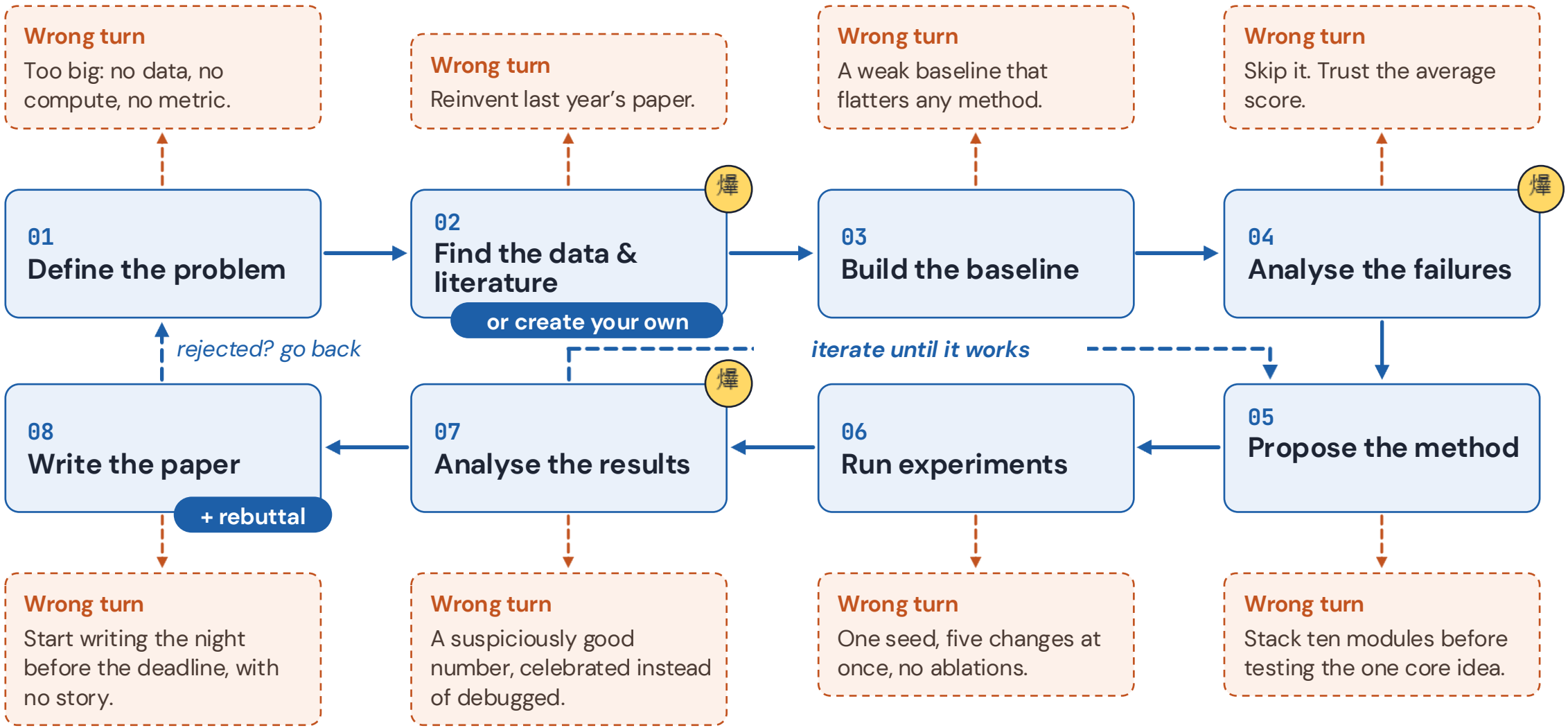
"Deeper GCN, worse results. Over-smoothing."

AI answers the questions you ask. Your knowledge pool decides which questions you can ask.

The research loop, and its wrong turns.

Blue = path · **Orange** = wrong turn

爆 **Look at your data**: at steps 02, 04 and 07, and whenever in doubt.



Step 01 up close: define the problem

Eight questions before you start.

01 Fits your PhD topic?

Each project should serve the big question. The topic can shift later, but on purpose, not by accident.

02 A new task, or an existing one? Both are hard, differently.

NEW TASK

You must convince people the task matters.

No data, no metric, no baselines: you often build all three.

Reviewers ask “why this task?” before “why this method?”

EXISTING TASK

Crowded field, strong and fast-moving state of the art.

You need a clear, real gain, not +0.3%.

High risk of being scooped, or called incremental.

03 Part of a series?

A PhD is 3–4 projects. Best if they line up into one thesis story.

P1 → P2 → P3 → **Thesis**

04 Where do you find it?

Related work, and the limitations and future-work sections of other papers. Reviewer complaints too.

05 Can you measure success?

A metric and a benchmark you trust. If you can’t measure it, you can’t claim it.

06 Is it interesting?

To you, first. Boring tasks kill motivation.

The worst research: the paper is exactly your first idea, with no obstacles. That idea was boring.

07 Is it feasible?

Data, compute, time. Training a huge model from scratch is not a PhD project.

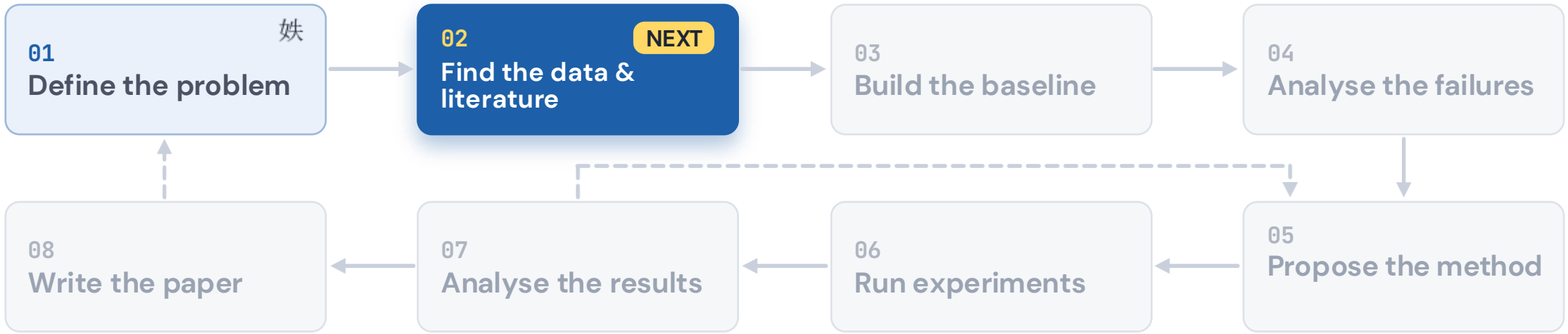
08 Where is the field going?

Embodied AI? Multimodal LLMs? Can your task connect to it, or add a modality? A PhD is one project; your tech stack is for life.

THE RESEARCH LOOP

Next: 02 · Find the data & literature

Look before you train. Verify before you trust.



Step 02 up close: data & literature

Look before you train. Verify before you trust.

The data

FIND

Where to look: Papers with Code, HuggingFace, the experiment tables of related papers.

Fit check: right task, right labels, enough samples, licence, standard split.

OBSERVE · don't use it straight away

Watch 50 random samples yourself, before any training.

Distributions: label imbalance, long tail, clip length, fps, resolution.

Label quality: noisy or ambiguous? Would two people agree?

ANTICIPATE

Hard parts: fast motion, occlusion, tiny differences between classes.

Shortcuts: could the background, camera or clip length give the answer away?

The literature

FIND

Search: Semantic Scholar, Google Scholar, Connected Papers. Follow citations both ways, Survey Paper.

LLMs give a fast overview, but check every paper actually exists.

VERIFY

Real or hype? Fair baselines, enough seeds, the standard split?

Reproducible? Code, weights, and open "can't reproduce" issues on GitHub.

ADAPT

Fits our data? Input format, modalities, clip length, compute cost.

Base model? Strong, maintained, easy to modify. Reproduce its number first.

Leave step 02 with: a dataset you understand + a baseline you can actually run

Step 02, plan B: create your own dataset

A dataset can be the contribution.

+ When it's worth it

No existing data fits, or it's tiny, biased or saturated.

A new task needs a new benchmark.

A good dataset defines a task and gets cited for years. There are venues just for it, like NeurIPS Datasets & Benchmarks.

- What it costs

Months of collection and annotation.

Ethics approval, consent, licences and privacy take time.

Reviewers ask: why not existing data? Is it big and diverse enough? Are the labels reliable?

HOW TO BUILD ONE

01

Define

Task, labels and metric first.

Written guidelines with examples.

02

Collect

Diverse people, scenes, views, conditions.

Consent, ethics, licence, privacy.

03

Annotate

Pilot round first, then scale.

AI pre-labels, humans verify. Measure agreement.

04

Benchmark

Split by subject or video, never by clip.

Run strong baselines on it.

05

Release

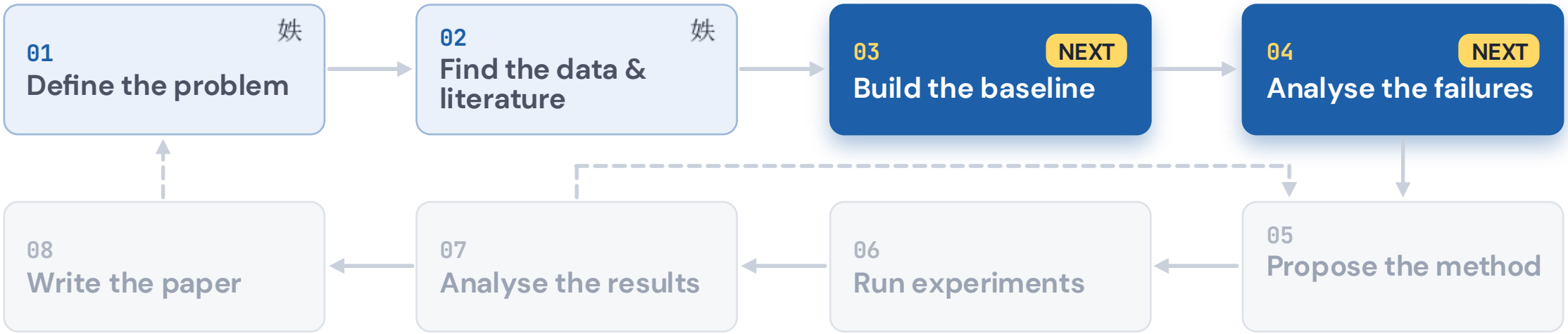
Datasheet, licence, code.

Host it, and keep maintaining it.

THE RESEARCH LOOP

Next: 03 + 04 · Build the baseline, find where it fails

Choose it like you're choosing the task.



Steps 03 + 04 up close: the baseline

Choose it like you're choosing the task.

刎 03 · Reproduce and dissect it

Can you reproduce the number? Same data, split, metric and seeds.

Where does the gain come from? A bigger backbone, more pretraining data, or the method itself?

Idea or trick? Test-time augmentation, ensembles, longer training, tuning on the test set.

Can't get close? Hidden details, or results that may not be reliable.

Can you code it from the paper alone? LLMs can do it too, but writing it yourself is how you really understand it.

Can you live with it? Clean code, affordable compute, easy to modify.

爆 04 · Find where it fails

Study the hard cases: which samples, classes and conditions break it?

Systematic or random? Only a systematic failure is worth a paper.

Room to grow? Oracle inputs, human agreement, seed noise: how far is the ceiling?

Is the benchmark healthy? Saturated, noisy, or gaps smaller than seed variance?

The failure you find becomes the motivation for your method in step 05.

HOW STRONG SHOULD THE BASELINE BE?



Too weak

+ Easy "state of the art". **–** Weak evidence: "why not compare with X?"

Sweet spot

Strong, recent, reproducible, failing in a clear way, with room above it.

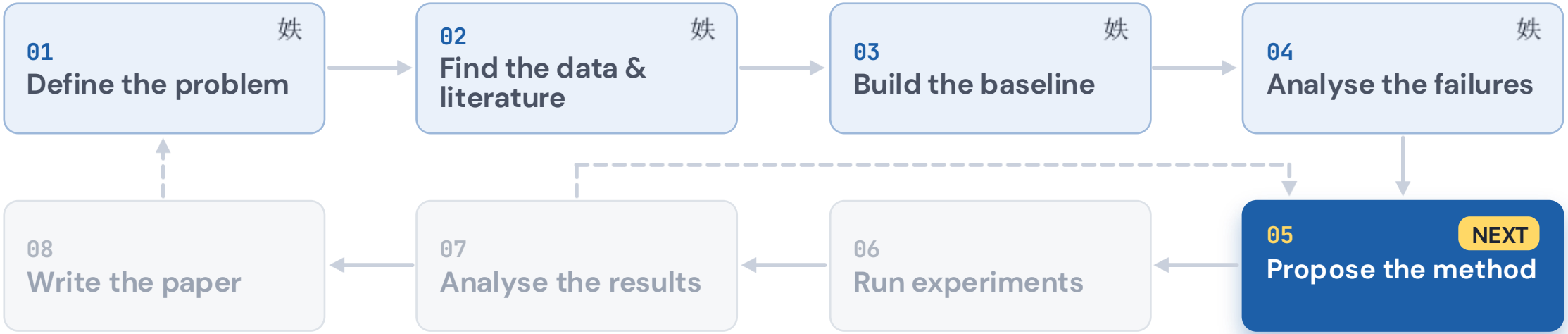
Too strong, saturated ▶

– Hard to move at all. **+** If you do, it's a breakthrough.

THE RESEARCH LOOP

Next: 05 · Propose the method

Three roads to a method.



Step 05 up close: propose the method

Three roads to a method.

爆 **Go bottom-up**: start from the failures you found in step 04, then design the fix. Not from a trendy module.

A + B = C

Combine existing ideas

What: module A from one line of work, B from another. The most common route.

Risk: "incremental", "just a combination".

`MAKE IT HOLD UP`

Show the glue: why the naive A + B fails, and what you changed.

Give an insight C reveals that A or B alone can't.

Ablate both parts; beat A + B′.

剣 Genuinely new

`THEORY-DRIVEN`

Start from a principle: a derivation, an objective, a bound. Explains *why*.

Watch out: do the assumptions hold on real data?

`DESIGN-DRIVEN`

Built for your problem's structure: time, skeletons, scoring rules.

Watch out: is it the design, or the tuning?

崽 Efficient, scalable, new

Efficient: same accuracy for less compute, memory or data. FlashAttention, LoRA, distillation.

Scalable: keeps improving as data and compute grow.

Brand-new architecture: the Transformer replaced RNNs. Highest risk, highest reward.

Watch out: compare at matched parameters and FLOPs.

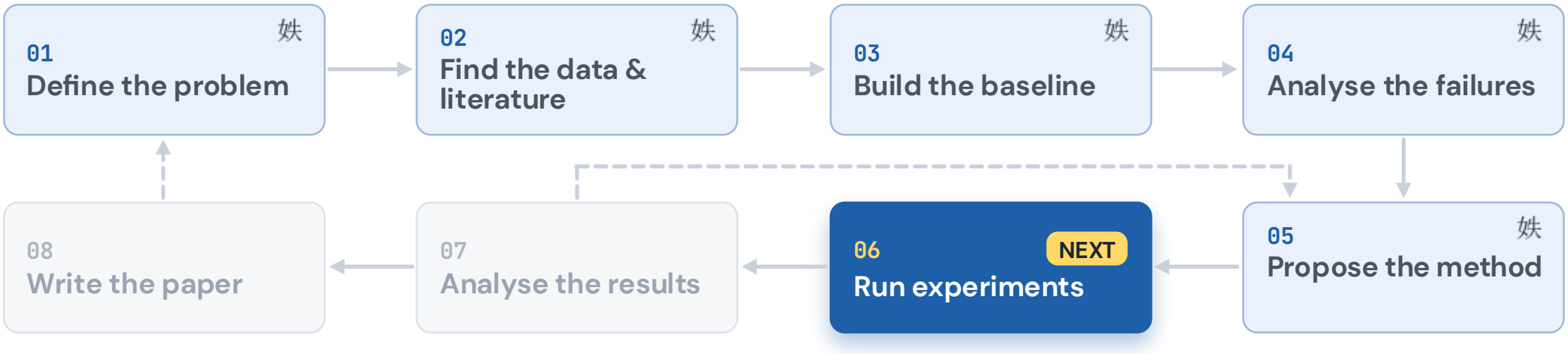
獞 **Benchmark saturated?** Stop chasing +0.1%. Change the axis: **few-shot** · **efficiency** · **new modalities** · **cross-domain** · **fine-grained** · **long-term temporal** · **interpretability** · **privacy** · **online learning** · **continual learning**

Any road: test the smallest version first. Novelty is the insight: what does it fix, and why does it work?

THE RESEARCH LOOP

Next: 06 · Run experiments

Every table answers a question.



Step 06 up close: run experiments

Every table answers a question.

億 Design the experiments

Main table: the latest and strongest methods, same split, same metric. Is last month's state of the art in it?

Comparisons: other datasets, backbones, settings. Does it generalise?

Module alternatives: swap your module for the obvious options. Why this design?

Ablation: remove one component at a time. Every claimed contribution gets a row.

Efficiency: parameters, FLOPs, memory, inference time vs accuracy.

Sensitivity: key hyperparameters. Robust or fragile?

Qualitative: visual examples, including where it still fails.

User study: when metrics can't judge quality, ask people.

憊 Good habits

#run	commit	change	expected	result	why?

Log every run in one table, with the columns above. Good or bad.

Predict before you run. Wrong predictions teach the most.

Analyse the wins too: which samples and classes improved, and why?

Know why it works: only then do you know what to push next, and what you can honestly claim.

Keep failed runs: negative results save you weeks later.

Back up your code: git commit every version, so any result traces back to the exact code.

Controlling variables is everything!!!

刻 Rigour

Controlled variables?
Same data, budget and tuning effort for you and the baselines.

Randomness controlled?
Seeds, data order, cuDNN determinism.

Repeated runs?
3–5 seeds, report mean ± std.

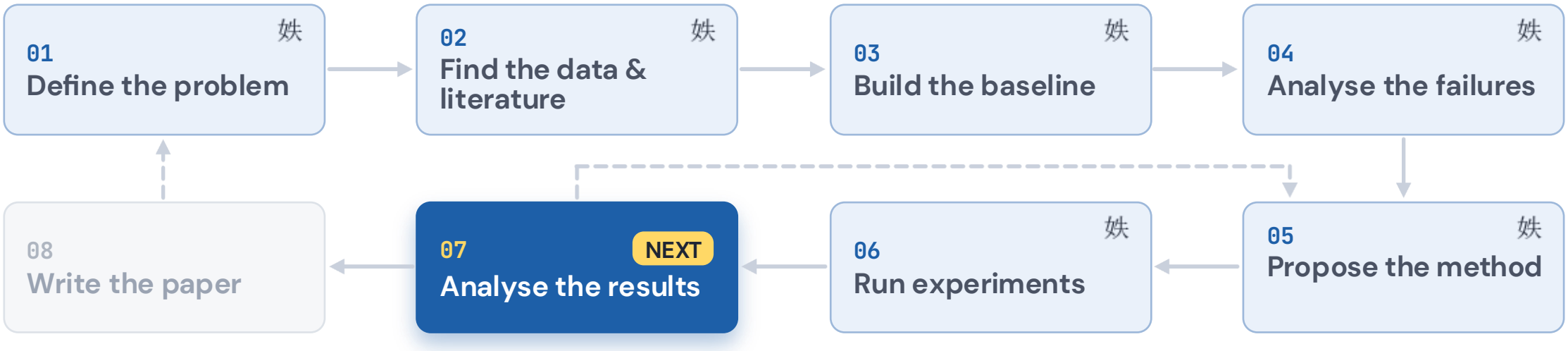
Significant?
t-test or bootstrap. Never tune on the test set.

Don't only ask why it failed. Ask why it worked. That tells you where to go next.

THE RESEARCH LOOP

Next: 07 · Analyse the results

Research is SGD: learn to read the gradients.



BEFORE STEP 07: A MINDSET

Research is stochastic gradient descent.

Your ideas are the gradients. Bad results often give you the biggest ones.

01

Surprise is the reward.

A surprising observation is the happiest moment a researcher can have.

02

−10 points is a signal.

Most people call it a dead experiment. Ask instead: what if we go the opposite way?

03

Flat is the worst.

Not better, not worse: zero gradient. The experiment told you nothing.

04

Every signal has a direction.

Negative? Go the other way. Positive? Push further.

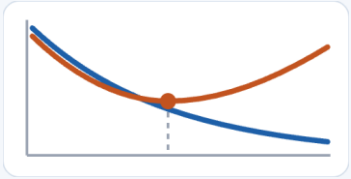
剣

Predict every experiment. Right? Keep extending the idea. Wrong? That's a surprise too: rethink your reasoning and explain why.

Step 07 up close: bad case analysis

🜄 Training curves

Train ↓, val ↑: overfitting. Stop at the dip; add data, augmentation, regularisation.



Both stay high: underfitting, or a bug. Overfit a single batch first.

Zig-zag: learning rate too high, batch too small, or noisy labels.

Spike or NaN: exploding gradients, FP16 overflow, log(0) or divide by zero. Try gradient clipping, BF16, a lower LR.

Flat from step one: LR too low, layers frozen, or gradients never reach the weights.

Loss ↓ but metric flat: the loss is not measuring what you care about.

Too good to be true: check for train/test leakage.

🜁 Statistics & visuals

Multiple seeds: mean ± std. A gain smaller than the noise is no gain.

Significance: paired t-test or bootstrap confidence interval.

t-SNE / UMAP: do the classes separate in feature space? Left: mixed. Right: clean clusters.



Confusion matrix: which classes get mixed up with which.

Grad-CAM / attention map / attention flow: is it looking at the right thing? Flow (rollout) traces attention through all layers.

Win / loss vs baseline: which samples you beat it on, and which you lose.

Regression tasks: predicted vs true scatter. Are extreme scores pulled to the mean?

🜂 **First ask: is it the model, or the data?** Dump the real input · Bucket errors by fps, length, resolution · Change only the preprocessing

Features & fusion: the quiet traps.

No error message. Just worse results.

Feature engineering

Scale mismatch: pixel coordinates next to unit-norm embeddings. Normalise each feature type.

Raw skeleton coordinates: centre on the hip, scale by the torso, or the model learns camera position.

Missing values as zeros: failed detections become "zero means class X".

Label leakage: a feature derived from the target, or from future frames.

Stale features: pre-extracted with a different backbone or resolution than you think.

Inconsistent augmentation: flip the video, then also swap left/right joints and sensor axes.

Multimodal fusion

Modality dominance: the easy modality wins and the others stop learning.

Fused < best single modality: misalignment or dominance, not "fusion doesn't help".

Dimension imbalance: 2048-d video next to 34-d pose. Project to the same size first.

Time misalignment: 25 fps video, 100 Hz IMU, 16 kHz audio, plus sync offsets.

Missing modality at test time: train with modality dropout.

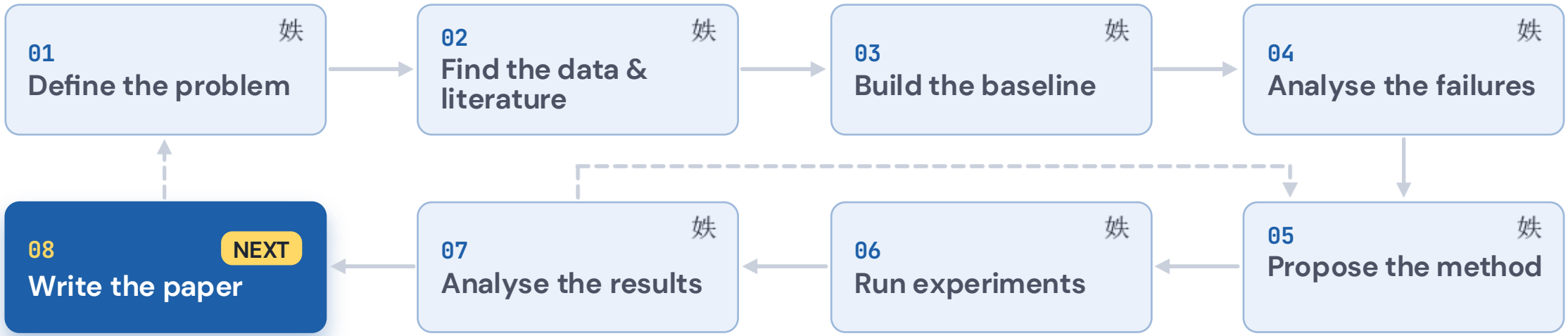
Cross-attention collapse: check how much attention each modality really gets.

Always report: each modality alone → a simple late fusion → then your fancy fusion

THE RESEARCH LOOP

Next: 08 · Write the paper

Reviewers look at the figures first.



Step 08 up close: write the paper

Reviewers look at the figures first.

圩 Story & intro

One storyline: problem → why it's hard → key insight → what we show.

Hook early: open with a concrete example, a striking failure or a number.

Contributions: three bullets, each backed by an experiment.

儢 Figures

Figure 1 tells the whole paper at a glance.

Reading order: left to right, top to bottom. One message per figure.

Muted colours: one colour per module, branch or method, used the same way everywhere.

Full layout: no empty gaps, fonts readable when printed.

珣 Equations

Define every symbol once, right where it first appears.

Consistent notation: bold vectors, capital matrices, one letter = one thing.

Check dimensions and indices. Does each equation match the code?

刊 Related work

Critical, not a list: what each line of work does, and what it misses.

Tie it back: end each paragraph with "unlike them, we …".

Cite the strongest and the latest, including competitors.

坲 Experiments

Each section answers a question: main result, ablation, analysis, failures.

Tables: best in bold, second underlined, ↑/↓ on metrics, same split.

Reproducible: mean ± std, all settings, code if you can.

Before you submit

Write the abstract and Figure 1 first, then the rest.

Play reviewer 2: what would you attack? Fix it now.

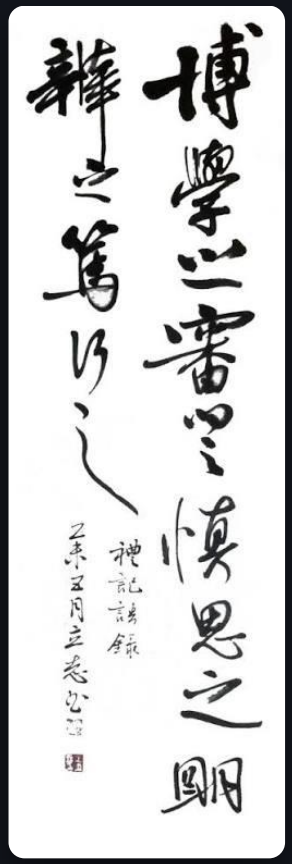
Always run an LLM check: notation, numbers vs tables, claims vs evidence, typos. It often surprises you.

Fresh eyes: a colleague too. You own the argument.

THANK YOU · QUESTIONS?

Thank you.

> **博学之，审问之，慎思之，**
> **明辨之，笃行之。——《中庸》**
>
> *Learn widely, question closely, think carefully, discern clearly, and act steadfastly. (The Doctrine of the Mean)*



Xu Dong University of Surrey · Gilbertines

嘁 dx199771.github.io/xudong

珣 github.com/dx199771

損 Google Scholar

愍 xd00101@surrey.ac.uk

吁 LinkedIn

圩 CV